UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sandi Holub

Obdelava velikih količin podatkov v oblaku

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2014

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sandi Holub

Obdelava velikih količin podatkov v oblaku

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Z rastjo obsega podatkov v zadnjih letih in še večjo predvideno rastjo v naslednjih letih se pojavi problem količine podatkov, ki jih je potrebno zajeti v obdelavah in analizah. Analize velikih količin podatkov so lahko infrastrukturno in časovno zelo zahtevne. Zaradi problematike obdelave velikih količin podatkov so se pojavile potrebe po učinkovitejši strojni infrastrukturi, z več računalniki povezanimi v gručo, ki pa si je podjetja ne morejo privoščiti ali pa je preprosto nočejo imeti in vzdrževati. Za odgovore na analitična vprašanja ali obdelave si lahko podjetja pomagajo z uporabo cenovno ugodnih in hitrih rešitev v oblaku.V okviru diplomskega dela najprej predstavite trenutno najbolj razširjen pojem pri hranjenju velike količine razpršenih podatkov v gručah Hadoop in njegovo orodje za obdelavo le teh MapReduce (YARN). Predstavite, kako si lahko podjetja pomagajo pri obdelavi velikih količin podatkov v oblaku (npr. Amazon). Izdelajte MapReduce aplikacijo, ki bo obdelala veliko količino podatkov po vaši izbiri. Primerjajte cenovni in časovni vidik obdelave takih podatkov v oblaku ali na lokalnem sistemu.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Sandi Holub, z vpisno številko **63050042**, sem avtor diplomskega dela z naslovom:

Obdelava velikih količin podatkov v oblaku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Aljaža Zrneca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 3. julija 2014 Podpis avtorja:

Ob tej priložnosti bi se rad zahvalil družini za vso podporo med študijem. Mojim soigralcem iz Ljubljana Silverhawks za zaupanje in čuvanje hrbta na tekmah. Pravtako bi se zahvalil mojemu delodajalcu in sodelavcem za potrpežljivost v obdobju pisanja diplome. Posebna zahvala gre moji partnerici, ki me je med pisanjem diplome spodbujala in mi stala ob strani. Na koncu bi se rad zahvalil še mentorju viš. pred. dr. Aljažu Zrnecu za korektno in strokovno mentorstvo pri izdelavi diplomske.

Kazalo

Seznam uporabljenih kratic

CERN (fra. Conseil Européen pour la Recherche Nucléaire) Evropska organizacija za jedrske raziskave

DNS *(angl. Domain Name System)* Sistem domenskih imen

EMR (*angl. Elastic MapReduce*) Tip Amazon strežnikov v oblaku za enostavno poganjanje gruč z MapReduce implementacijo

EUR (*angl. European Union Euro*) Valuta evro

GB (*angl. gigabyte*) Merska enota za količino podatkov v računalništvu

GDFS (*angl. Google Distributed File System*) Google implementacija distribuiranega datotečnega sistema

GPS (*angl. Global Postioning System*) Globalni sistem pozicioniranja ali sistem globalnega pozicioniranja, ki ga je zasnovalo ameriško obrambno ministerstvo

HDFS (*angl. Hadoop Distributed File System*) Apache Hadoop implementacija distribuiranega datotečnega sistema

HTTP (*angl. HyperText Transfer Protocol*) Protokol za prenos informacij po spletu

IDC (*angl. International Data Corporation*) Vodilni svetovni ponudnik za tržno inteligenco in svetovalne storitve na področju informacijske tehnologije, telekomunikacij in potrošnje

IP (*angl. Internet Protocol*) Komunikacijski protokol za prenašanje podatkov po omrežju

LTS (*angl. Long Term Support*) Označuje produkte, ki imajo neglede na novejše verzije še vedno podporo razvijalcev in se sproti posodabljajo

NDFS	(*angl. Nutch Distributed File System*) Prvotno poimenovanje HDFS v sklopu Nutch projekta
PB	(*angl. petabyte*) Merska enota za količino podatkov v računalništvu
RDBMS	(angl. Relational Database Management System) Relacijska podatkovna shramba
S3	(*angl. Simple Storage Solution*) Amazon enostaven datotečni sistem
SSD	(angl. Solid State Drive) Tip strojne opreme za shranjevanje podatkov, ki ne uporablja vrtečih diskov
SFTP	(*angl. Secure File Transfer Protocol*) Protokol za upravljanje datotečnega sistema na daljavo
SSH	(*angl. Secure Shell*) Varen protokol za upravljanje računalnika na daljavo, ki za varnost uporablja princip javnega in zasebnega ključa
SQL	(angl. Structured Query Language) Standardni jezik za pisanje poizvedb v podatkovnih shrambah
TB	(angl. terabyte) Merska enota za količino podatkov v računalništvu
ZB	(angl. zettabyte) Merska enota za količino podatkov v računalništvu
YARN	(angl. Yet Another Resource Negotiator) Komponenta Hadoop, ki skrbi za upravljanje z viri gruče
XML	(angl. Extensible Markup Language) Razširljiv označevalni jezik za izmenjavo strukturiranih dokumentov

Povzetek

Velika količina podatkov ali Big Data v svetu informatike pridobiva na prepoznavnosti. To so orodja, ki omogočajo shranjevanje in poizvedovanje po veliki količini podatkov. Hadoop je odprtokodni projekt podjetja Apache, ki združuje orodja za shranjevanje, obdelavo in poizvedovanje po strukturiranih ali nestrukturiranih podatkih. Podatke je potrebno obvladovati s pomočjo primerne infrastrukture, največkrat kar gruče računalnikov. Pri tem si lahko pomagamo z oblakom, če ne želimo imeti infrastrukture pri sebi. YARN (*angl. Yet Another Resource Negotiator*), MapReduce, Pig in HDFS (*angl. Hadoop Distributed File System*) so osnovne komponente projekta Hadoop in pripomorejo k enostavni implementaciji prve različice programske opreme. Z diplomsko nalogo si lahko bralec pomaga pri postavitvi osnovne gruče Hadoop in izdelavi Java aplikacije ali skripte Pig. Časovna in cenovna primerjava poganjanja v oblaku in lokalni gruči pa pomaga pri odločanju o nakupu infrastrukture.

Ključne besede: Hadoop, MapReduce, velika količina podatkov, gruča, Pig, oblak

Abstract

Big Data is gaining recognition in the world of information technology. These are tools that allow saving and retrieval of a large amount of data. Hadoop is an open source project of the company Apache, which combines the tools for storage, processing and retrieval of structured and or unstructured data. Data need to be managed with the help of appropriate infrastructure, which in most cases are clusters of computers. We can help ourselves using a cloud, if we do not wish to have the infrastructure nearby. YARN, MapReduce, Pig and Hadoop Distributed File System (HDFS) are the basic components of the Hadoop project and contribute to simple implementation of the first version of software. The reader can use this diploma thesis as help in setting up the basic Hadoop cluster and develop a Java application or Pig script. The time and price comparison of running in the cloud or local cluster can also help as decision-making process when buying infrastructure.

Keywords: Hadoop, MapReduce, large amount of data, cluster, Pig, cloud

Poglavje 1 Uvod

V zadnjih nekaj letih se je na področju informacijskih tehnologij razširil pojem Big Data ali velika količina podatkov. Kaj sploh je Big Data? Na splošno je to opis eksplozije podatkov zadnjih nekaj let, ki jih ustvarjajo ljudje, multimedijske naprave, senzorji ipd. To pušča za sabo sled vsega, kar se dogaja okoli nas. Če takšne podatke pravilno analiziramo, lahko pridemo do zelo natančnih vzorcev napovedovanja (tržišča, vedenja množice ipd.), ki jih podjetja lahko uporabijo za izboljšave poslovanja. Rast količine podatkov bo samo še večja, saj [6]:

- čedalje lažje in ceneje je ljudi in svet okoli nas opremiti s senzorji in napravami, ki zbirajo podatke [6],
- stalni napredek v znanosti in tehnologijah računalniškega učenja olajša rudarjenje pomembnih informacij iz podatkovnih tokov [6],
- podjetja so se začela zavedati uporabnosti senzorskih podatkov in računalniškega učenja za izboljšanje poslovanja in napovedovanja trendov [6].

Orodja za obdelavo in hranjevanje velike količine podatkov morajo v osnovi zagotavljati naslednje:

- hranjenje velike količine podatkov,
- paralelno izvajanje poizvedb v računalniških gručah,
- visoke hitrosti poizvedovanja,
- programska ogrodja za pisanje poizvedb,
- prikazovanje rezultatov obdelav končnim uporabnikom.

Pojem Big Data ne opisuje samo količine podatkov in infrastrukture, ampak tudi algoritme za odgovarjanje na analitična vprašanja. Znanja iz podatkovnega rudarjenja, računalniškega učenja in najnovejših algoritmov za iskanje povezav in trendov so ključnega pomena pri uspešni implementaciji projektov Big Data.

Podatki so povsod okoli nas. Če smo včasih lahko s fotoaparatom naredili nekaj deset slik, preden smo napolnili film, jih sedaj lahko naredimo veliko več s telefonom. Če jih delimo na spletu, postanejo del ogromnih spletnih skladišč. Okoli nas je vse več senzorjev, ki svojim lastnikom pošiljajo podatke. Meteorološke postaje, vodovodni števci, kartice ugodnosti, telefoni, GPS (*angl. Global Positioning System*) in navsezadnje tudi internet, vse to so senzorji, ki nas obkrožajo v vsakodnevnem življenju [1].

Večina podatkov, ki nastanejo, so nestrukturirane narave. To so tekstovne datoteke, slike, videi, internetni klepeti, elektronska pošta itd. Lahko bi rekli, da so nestrukturirani podatki vsi podatki, ki nimajo opredeljene strukture (npr. tabela v podatkovni bazi) [1].

Lahko rečemo, da velike količine podatkov prihajajo iz zelo različnih virov in področij, v razmislek si lahko pogledamo naslednje primere:

- "New York Stock Exchange" vsak dan ustvari en terabajt novih podatkov o trgovanju z delnicami [1].
- "Facebook" shranjuje več kot 10 milijard slik, ki skupaj zasedajo približno en PB (*angl. petabyte*) [1].
- Spletna stran Ancestry.com shranjuje približno tri PB podatkov [1].
- "The Internet Archive" shranjuje približno dva PB podatkov, ki se vsak mesec razširijo za dvajset TB (*angl. terabyte*) [1].
- Ženevski pospeševalec delcev je v treh letih ustvaril 75 PB podatkov in razširil količino podatkov v CERN-ovem (*fra. Conseil Européen pour la Recherche Nucléaire*) obvladovanju na 100 PB [1].

Večina podatkov je zaprtih v spletnih iskalnikih, znanstvenih in finančnih institucijah. Ali pojem velike količine podatkov sploh vpliv na manjše organizacije ali celo na posameznike? Seveda, količina digitalnih podatkov, ki jih ustvarjajo posamezniki, močno narašča [1]. Če ste že kdaj kupovali v spletnih trgovinah, ste morda opazili, da oglasi, ki jih prejemate med brskanjem po spletnih straneh, postanejo prilagojeni vašim nakupom (Google adds, Facebook). Microsoft MyLifeBits je eksperiment, ki poskuša hraniti vse podatke o posamezniku (od dokumentov, prezentacij, pošte, slik, videov in celo telefonskih pogovorov).

Si lahko predstavljate, da bi v prihodnosti lahko imeli nad temi podatki hiter in pameten iskalnik podoben Googlu?

Z rastjo količine podatkov v zadnjih letih in predvideno še večjo rastjo v prihodnjih letih se pojavi problematika količine podatkov, ki jih je potrebno zajeti v obdelavah in analizah. Analize in obdelave na velikih količinah podatkov so lahko infrastrukturno in časovno zelo zahtevne [1].

Ni lahko predvideti celotne količine podatkov shranjenih v elektronski obliki. Raziskava IDC (*angl. International Data Corporation*) je ocenila količino podatkov na 1.8 ZB (*angl. zettabyte*) do leta 2011 in 8.6 ZB do 2015. Na podlagi teh številk se bo do leta 2020 količina podatkov povečala na vsaj 40 ZB [1,10].

Zaradi problematike obdelovanja velikih količin podatkov so se pojavile potrebe po boljši in hitrejši infrastrukturi z več računalniki, povezanih v gručo, ki pa si je podjetja ne morejo privoščiti ali pa je preprosto nočejo imeti in vzdrževati. Za odgovore na analitična vprašanja ali obdelave si lahko podjetja pomagajo do cenovno ugodnih in hitrih rešitev v oblaku.

Tako kot je predvidena rast količine podatkov, je predvidena rast uporabe oblaka. IDC je predvidela, da bo do leta 2020 čez oblak na tak ali drugačen način potovalo 37 % vseh podatkov [1,10].

V okviru diplomskega dela najprej predstavim trenutno najbolj razširjen pojem pri hranjenju velike količine distribuiranih podatkov v gručah Hadoop in njegovo programsko ogrodje za obdelavo le-teh MapReduce. Za razvoj MapReduce aplikacije sta uporabljena razvojno okolje Eclipse in programski jezik Java.

Za predstavitev delovanja MapReduce s pomočjo Hadoop razvijem aplikacijo MapReduce za iskanje povprečne temperature iz javno dostopnih podatkov vremenskih postaj po Severni Ameriki.

Lokalno razvito aplikacijo poženem na Amazonovi gruči in predstavim cenovni in časovni vidik poganjanja v oblaku ali lokalnem okolju. Izbrana infrastruktura je Amazon EMR (*angl. Elastic MapReduce*) in S3 (*angl. Simple Storage Solution*) datotečni sistem za hranjenje podatkov in metapodatkov. Za boljšo primerjavo časov izvedbe obdelav gruč v oblaku in lokalne infrastrukture predstavim način povezovanja na oddaljena vozlišča v oblaku in obdelave poženem nad podatki, ki so shranjeni na lokalnem HDFS vozlišč. Nekatere vmesne korake prikažem s Pig Latin jezikom, ki je bolj analitične narave in bralcem iz relacijskega sveta podatkovnih baz bolj razumljiv.

Poglavje 2 Hadoop

Apache Hadoop je odprtokodni projekt, ki združuje orodja za shranjevanje in paralelno obdelavo velike količine distribuiranih podatkov v računalniških gručah. Glavni gradniki Hadoop so HDFS, YARN in MapReduce. HDFS je Hadoop implementacija tehnologije shranjevanja podatkov po principu distribuiranega datotečnega sistema. YARN skrbi za dodeljevanje, nadziranje, upravljanje računalniških virov ter paralelizem uporabniških aplikacij vozlišč v gruči. MapReduce je programski model in implementacija programskega ogrodja za obdelavo podatkov [11].

Preden v naslednjih poglavjih opišem osnovne gradnike Hadoop, si poglejmo kratko zgodovino.

Leta 2002 sta Doug Cutting in Mike Cafarella pri Apache dobila nalogo, da iz takrat zelo razširjene odprtokodne zbirke algoritmov tekstovnega iskalnika Apache Lucene razvijeta spletni iskalnik. Apache Nutch, kot sta ga poimenovala, je že isto leto dobil delujoči spletni iskalnik in pripadajočega pajka za indeksiranje spletnih strani. Med razvojem sta ugotovila, da bi bila strežniška arhitektura za uporabo Nutch na svetovnem spletu predraga. Takrat je bila strežniška arhitektura, ki bi podpirala eno milijardo indeksiranih spletnih strani, ocenjena na približno pol milijona dolarjev. Vzdrževanje take arhitekture pa približno trideset tisoč dolarjev na mesec. Odločili so se za nadaljevanje razvoja, saj so bili mnenja, da bi odprtokodna rešitev spletnega iskalnika veliko pripomogla in širši množici omogočila dostop do algoritmov spletnih tehnologij. Pomoč je prišla s strani Googla, ki je leta 2003 objavil članek, ki je vseboval razlago konceptov arhitekture distribuiranega datotečnega sistema. Članek je opisoval GDFS (*angl. Google Distributed File System*), ki je bil takrat pri Googlu v produkcijski uporabi [15]. Pri Apache so bili mnenja, da bi podobna rešitev odpravila težave z drago arhitekturo in vzdrževanjem. Leta 2004 so začeli z razvojem odprtokodne rešitve distribuiranega datotečnega sistema NDFS (*angl. Nutch Distributed File System*). Leta 2004 je Google v članku razkril koncept MapReduce [9]. Leta 2005 so pri Apache Nutch imeli delujočo arhitekturo MapReduce v svoji implementaciji distribuiranega datotečnega sistema. Še istega leta so vse algoritme prilagodili MapReduce modelu. Kmalu so ugotovili, da je uporaba takega sistema presegla spletne iskalnike in bi lahko bila uporabljena na različnih področjih, kjer je potrebna obdelava velike količine podatkov. Tako je leta 2006 nastal projekt

Apache Hadoop. Uporabnost Hadoop so videli pri Yahoo! in še istega leta je Doug Cutting iz Apache prešel k Yahoo!, kjer je dobil ekipo, zadolženo za razvoj Hadoop. Nutch Distributed File System so preimenovali v Hadoop Distributed File System. Zanesljivost Hadoop je Yahoo! potrdil leta 2008, ko je javnosti razkril, da za generiranje svojih indeksov zaupa deset tisoč procesorskim jedrom povezanih v Hadoop gručo. Leta 2009 je Yahoo! podrl rekord v sortiranju enega terabajta podatkov. Sortiranje je trajalo 62 sekund. Prejšnji rekord je bil postavljen pri Google in sicer 68 sekund. Oba rekorda sta bila postavljena na zelo dragi arhitekturi. Ocenjeno je bilo, da je Yahoo! podrl rekord na arhitekturi, ki stane okoli pet milijonov dolarjev. Leta 2012 je rekord znova padel. Pri Googlu so pognali Google Computing Engine, ki je Googlova rešitev za obdelavo velikih količin podatkov v oblaku, in za pičlih 9 dolarjev rekord postavili na 54 sekund. Uporabljenih naj bi bilo 4012 procesorskih jeder in 1003 fizičnih diskov [12]. Hadoop je sedaj najbolj poznan pojem, ko gre za distribuirane datotečne sisteme in obdelave velikih količin podatkov. Za zaključek kratke zgodovine še razlaga izvora imena in logotipa (glej sliko 1), ki jo je podal avtor Doug Cutting, prevedena iz angleščine: »Ime Hadoop ni kratica. Ime je povsem izmišljeno. Ime je moj sin dal svojemu rumenemu plišastemu slonu. Je kratko, lahko ga je črkovati in izgovarjati, je brez posebnega pomena in ni bilo še nikjer uporabljeno. To so moji kriteriji poimenovanja. Otroci so dobri v izmišljanju takih imen. Prav tako je Google izraz, ki ga uporabljajo otroci.« [1]

Slika 1: Hadoop logotip [8]

2.1 HDFS

HDFS je distribuiran datotečni sistem, oblikovan in prilagojen shranjevanju velikih količin podatkov v cenovno dostopni strežniški gruči. HDFS ima strežnik/odjemalec arhitekturo. HDFS podpira klasično hierarhično organizacijo datotek, ki so lahko shranjene v večnivojske mape. Podobno kot pri drugih datotečnih sistemih lahko uporabnik dodaja, premika, briše in preimenuje datoteke ali mape. V arhitekturi HDFS (glej sliko 2) je datoteka razdeljena na enega ali več enako velikih blokov (velikost bloka lahko nastavimo v nastavitvah HDFS), ki so shranjeni na različnih podatkovnih vozliščih. Poleg delitve datotek na bloke lahko za vsak blok določimo število kopij, ki jih hranimo v HDFS. Gruča HDFS je sestavljena iz enega ali več glavnih vozlišč (*angl. Name Node*) (več v primeru zagotavljanja razpoložljivosti sistema ob izpadu glavnega vozlišča) in enega ali več podatkovnih vozlišč (*angl. Data Node*). Tipična namestitev HDFS vsebuje eno glavno vozlišče in več podatkovnih vozlišč. Čeprav ni

omejitve, koliko procesov podatkovnih vozlišč lahko namestimo na eno vozlišče, se je v praksi dobro držati pravila – eno podatkovno vozlišče na vozlišče v gruči [8].

Naloge glavnega vozlišča [8]:

- upravlja dostop do podatkov,
- skrbi za odpiranje, zapiranje in preimenovanje datotek in map,
- določa, na katerih podatkovnih vozliščih bodo posamezni bloki datotek,
- določa replikacijo blokov datotek,
- shranjuje metapodatke HDFS gruče (replikacije, lokacije blokov ipd.).

Naloge podatkovnega vozlišča [8]:

- upravlja shrambo na vozliščih, v katerih je nameščen,
- shranjuje, briše in replicira bloke datotek na zahtevo glavnega vozlišča,
- skrbi za operacije branja in pisanja.

2.1.1 Obvladovanje odpovedi strojne opreme

Zlasti pri velikih gručah je vedno katero od vozlišč nedelujoče. Zaradi tega je ključna funkcionalnost HDFS zagotavljanje hitrega zaznavanja in samodejno odpravljanje napak. Pri izpadih gre za izpade glavnega ali podatkovnega vozlišča. Podatkovno vozlišče v časovnih intervalih pošilja statusne signale glavnemu vozlišču. Če glavno vozlišče prejme signal, je podatkovno vozlišče delujoče. V primeru, da glavno vozlišče ne prejme signala v določenem intervalu, označi podatkovno vozlišče za nedelujoče (*angl. Dead Node*). Podatki na nedelujočih podatkovnih vozliščih niso na voljo, prav tako glavno vozlišče ne pošilja več zahtev po branju in pisanju podatkovnem vozlišču. Nedelovanje podatkovnega vozlišča povzroči padec števila replikacij blokov v gruči. V tem primeru glavno vozlišče identificira bloke, ki so bili shranjeni na nedelujočem podatkovnem vozlišču in določi novo vozlišče, na katerega replicira bloke. Ko se odpravi napaka na podatkovnem vozlišču in glavno vozlišče dobi prvi delujoč statusni signal, znova vzpostavi optimalno stanje replikacij. Pri izpadu glavnega vozlišča je potreben administratorski poseg v primeru, da gruča nima sekundarnega glavnega vozlišča. Če sekundarno glavno vozlišče obstaja, potem ta prevzame gručo, dokler se napaka ne odpravi [8].

2.1.2 Velika količina podatkov

HDFS omogoča aplikacijam dostop do velikih količin podatkov. Tipična velikost datoteke se meri v giga ali tera bajtih. HDFS je optimiziran za shranjevanje velikih datotek. Arhitektura omogoča enostavno dodajanje podatkovnih vozlišč v gručo za zagotavljanje razširljivosti po potrebi. Podpira več sto milijonov datotek. Tipične aplikacije, ki uporabljajo HDFS, zahtevajo velike hitrosti branja, ki jih taka arhitektura omogoča, saj bloke datotek bere v paralelnem načinu iz več različnih podatkovnih vozlišč. [8]

2.1.3 Zagotavljanje konsistence podatkov

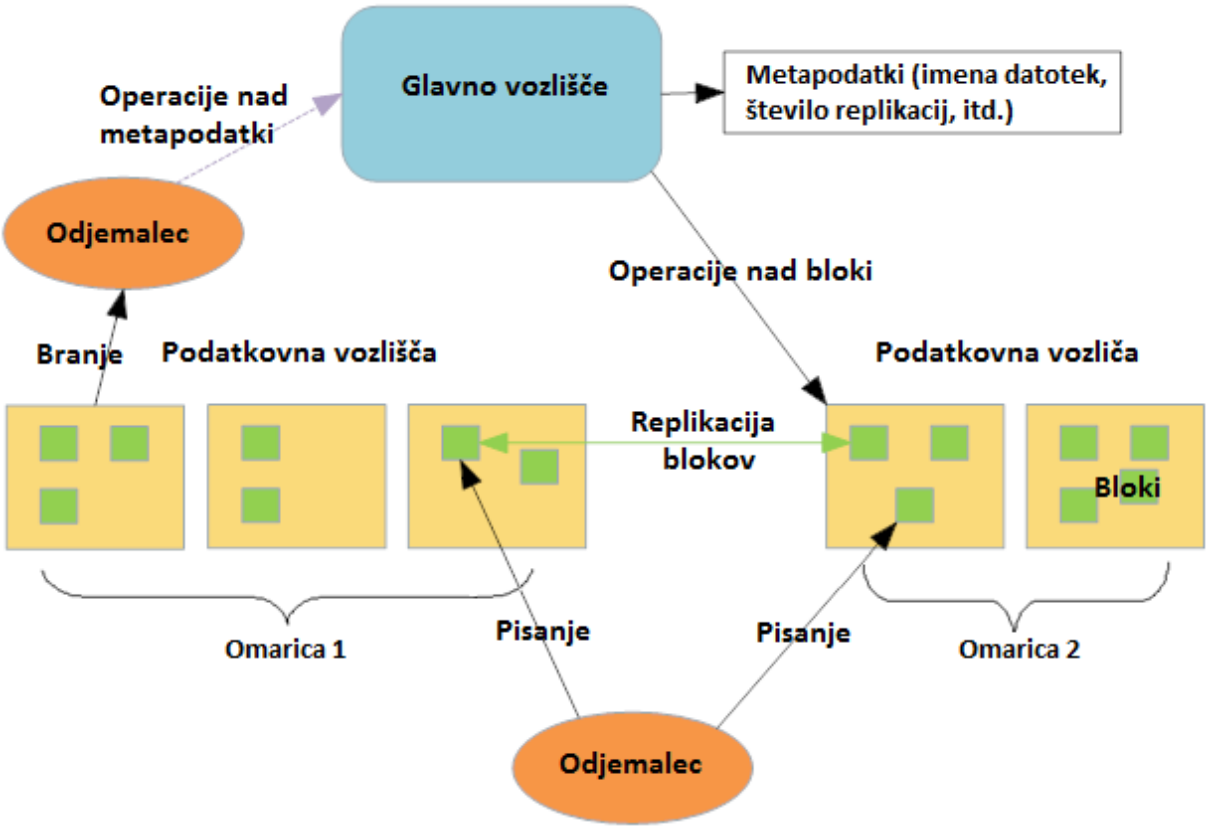
Zaradi narave distribuiranega shranjevanja podatkov je zelo pomembno zagotavljanje enakosti podatkov po celem sistemu. Podatkovni model HDFS je zgrajen s predpostavko, da se podatek oz. datoteka enkrat shrani in velikokrat bere. Ni namenjen dodajanju vsebine izvornim datotekam. Ta predpostavka močno olajša zagotavljanje konsistence podatkov med obdelavami in zagotavlja visoko hitrost branja. [8]

2.1.4 Poganjanje aplikacij v bližini podatkov

Obdelave so veliko bolj učinkovite, če se izvajajo v bližini podatkov, nad katerimi delajo. To je predvsem pomembno, če je količina podatkov zelo velika. Zaradi tega se zmanjša pretok podatkov po omrežju in poveča prepustnost celotnega sistema. HDFS predpostavka je, da je eno veliko obdelavo bolje razdeliti na manjše obdelave in jih pognati na vozliščih, kjer podatki so. HDFS že sam po sebi vsebuje vmesnike za premikanje obdelav bližje podatkom in uporabnikom za to ni potrebno skrbeti. [8]

2.1.5 Združljivost z različnimi operacijskimi sistemi in strojno opremo

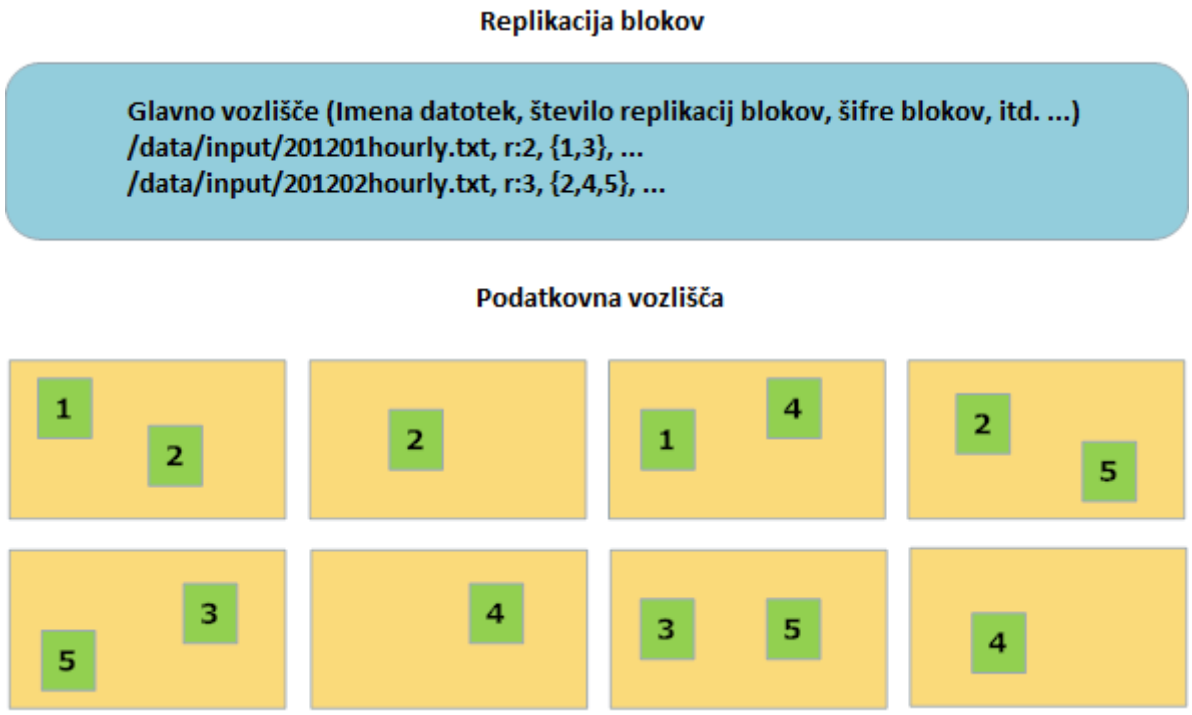
Arhitektura HDFS je narejena tako, da omogoča prenosljivost med različnimi operacijskimi sistemi. HDFS je napisan v programskem jeziku Java. Vse nastavitve so v XML (*angl. Extensible Markup Language*) datotekah. Zaradi tega je Hadoop na sploh primeren za ponudnike storitev v oblaku, saj je namestitev gruče preko XML datotek obvladljiva in enostavno programsko rešena. [8]



Slika 2: Prikaz arhitekture HDFS [8]

2.1.6 Replikacija blokov

HDFS vsako datoteko shrani kot zaporedje enako velikih blokov. Za zagotavljanje konsistence podatkov ob odpovedih strojne opreme je vsak blok datoteke shranjen v določenem številu kopij. Za vsako datoteko lahko poljubno nastavimo število kopij blokov. Število kopij se lahko nastavi ob kreiranju datoteke ali pa kasneje, ko je datoteka že shranjena. Glavno vozlišče vsebuje metapodatke o številu kopij posameznih blokov. Signali, ki jih pošiljajo podatkovna vozlišča, vsebujejo med drugim tudi informacije o repliciranih blokih, ki jih upravlja (glej sliko 3) [8].



Slika 3: Prikaz replikacije blokov na podatkovna vozlišča [8]

2.1.7 Izbira lokacije repliciranega bloka

HDFS se od drugih distribuiranih datotečnih sistemov najbolj razlikuje v svojih algoritmih za optimizacijo shranjevanja kopij datotečnih blokov. Pomembno je, da se distribuiran datotečni sistem zaveda topologije sistema. HDFS poleg razporejanja blokov na podatkovna vozlišča upošteva še, v kateri strežniški omari je vozlišče (*angl. Rack-awareness*). To pripomore k večji zanesljivosti, razpoložljivosti podatkov ter optimizira porabo pasovne širine omrežja. V večjih gručah HDFS mora komunikacija med omaricami potekati preko visoko zmogljivih stikal. V večini primerov je omrežje znotraj ene strežniške omare hitrejše kot med omarama. Glavno vozlišče za vsako podatkovno vozlišče pozna lokacijo omare. Tako lahko HDFS distribuira kopije blokov na podatkovna vozlišča v različnih omaricah. To pa pripomore k zanesljivosti in razpoložljivosti podatkov v primeru izpada celotne strežniške omare. Poleg tega lahko HDFS pri operaciji branja uporabi pasovno širino več omar hkrati. Tak algoritem prinese počasnejše pisanje. Če vzamemo primer replikacijskega faktorja, tri in dve strežniški omarici. HDFS bo za optimizacijo pisanja shranil en blok v prvo podatkovno vozlišče v prvi omarici, drugega v drugo vozlišče v prvi omarici in tretjega v prvo vozlišče v drugi omarici. Poleg algoritmov za shranjevanje kopij blokov vsebuje HDFS še algoritme za izbor blokov ob operaciji branja. Za zmanjšanje časov branja čez omrežje HDFS izbere blok, ki je najbližje vozlišču, ki je zahtevalo blok. [8]

2.1.8 Dostopanje do HDFS

Do HDFS lahko dostopamo na različne načine. Poleg knjižnic Java in C se lahko priklopimo tudi preko brskalnika HTTP (*angl. HyperText Transfer Protocol*). Na voljo imamo ukaze v lupini za delo s podatki in administracijo datotečnega sistema. Ukazi za delo s podatki so podobni ukazom v razširjenih lupinah (npr. bash, csh). [1]

- -cat
- -chmod
- -chown
- -df
- -cp
- -copyFromLocal
- -copyToLocal
- -mkdir
- -ls
- -rm
- -tail

Tabela 1 prikazuje primere poganjanja HDFS ukazov za upravljanje z vsebino datotečnega sistema.

Akcija	Ukaz
Ustvari mapo primer	hsingle@hsingle:~$ hdfs dfs –mkdir /primer
Poglej vsebino datoteke primer.txt	hsingle@hsingle:~$ hdfs dfs –cat /primer/primer.txt
Kopiraj datoteko primer.txt v primer2.txt	hsingle@hsingle:~$ hdfs dfs –cp /primer/primer.txt /primer/primer2.txt
Izbriši datoteko primer.txt	hsingle@hsingle:~$ hdfs dfs –rm /primer/primer.txt

Tabela 1: Primeri ukazov za upravljanje z vsebino datotečnega sistema HDFS

Ukazi za administracijo HDFS [1]

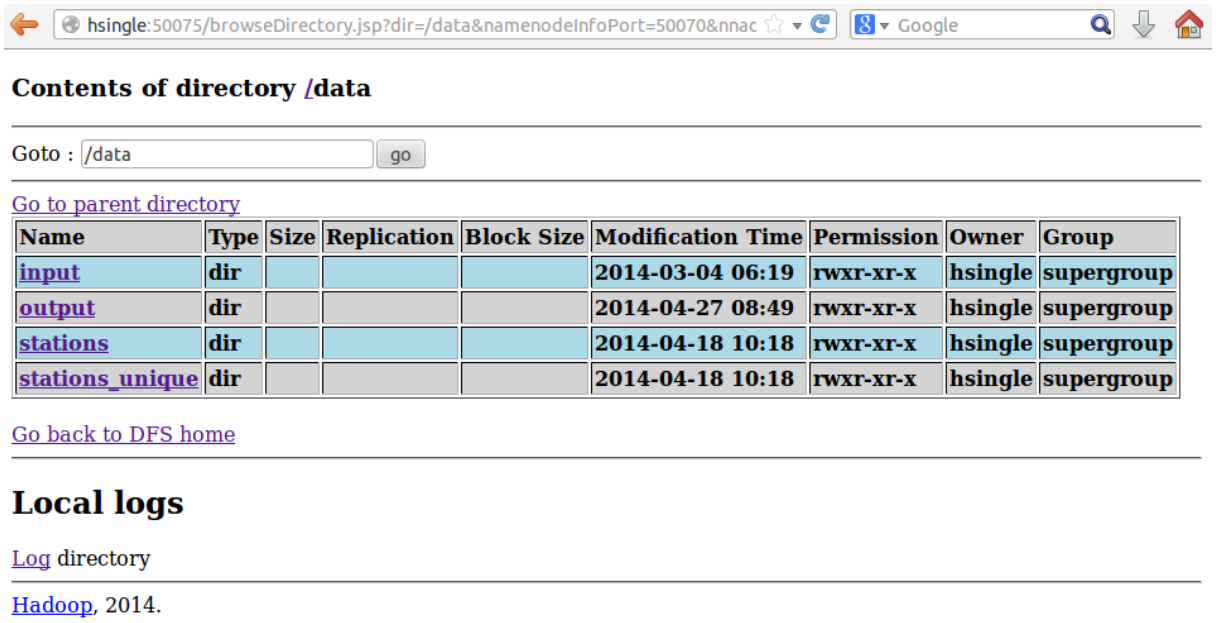
- namenode –format
- namenode
- datanode,
- dfsadmin

Tabela 2 prikazuje primere poganjanja HDFS ukazov za administracijo datotečnega sistema.

Akcija	Ukaz
Postavitev gruče v varni način	hsingle@hsingle:~$ hdfs dfsadmin –safemode enter
Poglej seznam DataNode v gruči	hsingle@hsingle:~$ hdfs dfsadmin –report
Izločitev DataNode1 iz gruče	hsingle@hsingle:~$ hdfs dfsadmin –decommission DataNode1

Tabela 2: Primeri ukazov za administracijo datotečnega sistema HDFS

Podatke v HDFS lahko pregledujemo preko spletnega brskalnika na privzetih vratih 50075 (glej sliko 4).



Slika 4: Prikaz HTTP vmesnika za pregledovanje datotek v HDFS

2.2 YARN in MapReduce 2.0

MapReduce je programski model za obdelavo in generiranje velikih količin podatkov s pomočjo algoritmov optimiziranih za paralelno in distribuirano delovanje v gruči. MapReduce lahko deluje nad podatki shranjenimi v nestrukturirani obliki na podatkovnem sistemu HDFS ali v strukturirani obliki v podatkovni shrambi. Programski model MapReduce je bil razvit in poimenovan pri Googlu. Najbolj razširjena odprtokodna implementacija je uporabljena v Apache Hadoop [1].

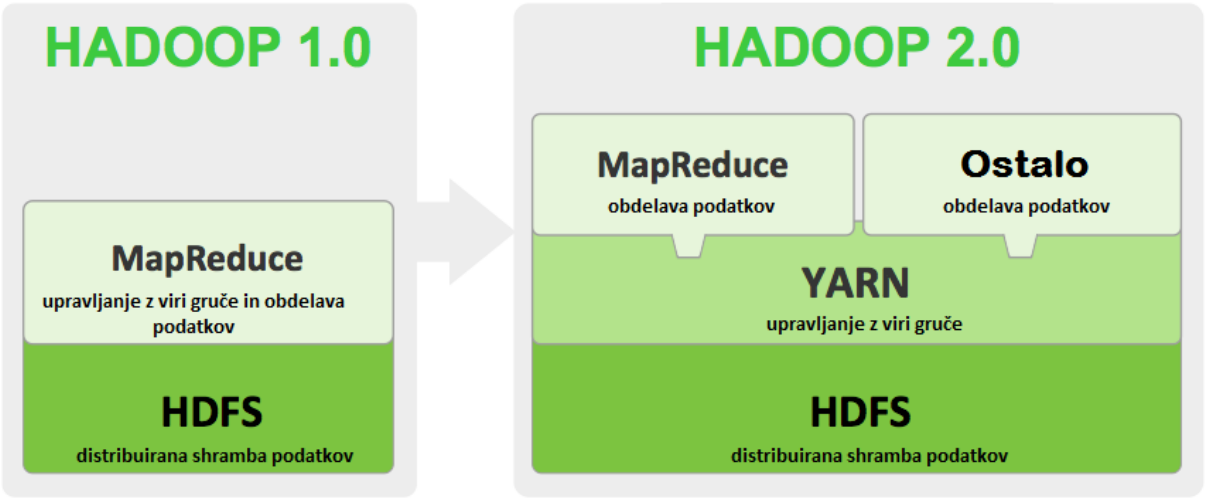
YARN je v Apache Hadoop od verzije 2.x.x naprej prenovljena implementacija MapReduce component, ki skrbijo za izvajanje, nadziranje in upravljanjem z računalniškimi viri obdelav v gruči. V prejšnjih verzijah (glej sliko 5) je za upravljanje računalniških virov skrbel proces znotraj MapReduce t. i. sledilnik obdelav (*angl. Job Tracker*). MapReduce se s pomočjo YARN lahko osredotoča na to, kar zna najbolje, obdelavo podatkov. YARN poskuša izrabiti koncepte lokalnosti podatkov, obdelave dodeljuje vozliščem, ki so najbližje podatkom [5].

Glavna ideja YARN je, da dve glavni funkcionalnosti starega sledilnika obdelav upravljanja računalniških virov in spremljanje obdelav razdeli na dve ločeni aplikaciji. Na globalnega upravitelja virov (*angl. Resource Manager*) in na enega upravitelja aplikacij (*angl. Application Master*) na obdelavo. Poleg teh dveh komponent je na vsako vozlišče nameščen še upravitelj vozlišča (*angl. Node Manager*), ki skrbi za izvajanje obdelav na vozlišču [1,5].

2.2.1 Arhitektura YARN

Če povzamemo poglavje 3.2 in nadaljujemo s podrobnejšo razlago nalog posamezne komponente YARN:

Upravitelj virov je t. i. dirigent YARN vozlišč. Skrbi za spremljanje obdelav in razporejanje računalniških virov. V osnovni konfiguraciji gruče je predviden en upravitelj virov. Na vsako podatkovno vozlišče je nameščen en upravitelj vozlišča, ki skrbi za poganjanje obdelav na vozlišču. Poleg poganjanja obdelav upravitelj vozlišča skrbi za pošiljanje informacij o statusu vozlišča upravitelju virov. Za vsako obdelavo upravitelj virov izbere eno vozlišče in na njem se požene upravitelj aplikacij. Ta postane skrbnik za obdelavo in skrbi za komunikacijo med drugimi vozlišči, izbranimi za obdelavo. Upravitelj aplikacij posreduje vse potrebne informacije o obdelavi upravitelju virov. Ko se obdelava zaključi, se upravitelj aplikacij ugasne. Slika prikazuje razliko v arhitekturi Hadoop 1.0 in 2.0 [1,5].



Slika 5: Primerjava logične arhitekture Hadoop 1.0 in Hadoop 2.0 [7]

2.2.2 Delovanje MapReduce programskega modela

MapReduce je v osnovi razdeljen na dva koraka: preslikaj (*angl. map*) in skrči (*angl. reduce*) [1,11]. Oba imata vhodni parameter in rezultat zapisan kot ključ in vrednost. Tip podatka programer poljubno izbere. Programer mora implementirati funkcijo preslikovanja in krčenja [1]. Za primer vzemimo naslednje vhodne vrstice v aplikacijo MapReduce, ki računa povprečno temperaturo po letih (izsek iz podatkov vremenskih postaj opisanih v tretjem poglavju):

WBAN,LETO MESEC DAN,URA,TEMPERATURA
03852,20130112,0655, 15
03011,20120101,0015,-5
03091,20120124,0940, 3
03712,20130215,0925, -2
04223,20130123,0440, 23

Funkcija preslikave je v tem primeru enostavna. Iz vrstic je potrebno izluščiti letnico in temperaturo. Preslikovanje služi kot transformacija vhodnih podatkov v obliko, s katero bo lahko krčenje hitro izračunalo povprečno temperaturo po letih. Preslikovanje je pravo mesto za logiko izločanja slabih vrstic npr. manjkajoči podatki, napake v obliki vrstice, temperature označene kot slab podatek ipd. Vhod v preslikovanje je v obliki ključ-vrednost, ki vsebuje številko vrstice kot ključ in vrstico kot vrednost [1].

KLJUČ, VREDNOST
(102, »03852,**2013**0112,0655, **15**«)
(12, »03011,**2012**0101,0015,**-5**«)
(23, »03091,**2012**0124,0940, **3**«)
(152, »03712,**2013**0215,0925, **-2**«)
(231, »04223,**2013**0123,0440, **23**«)

Ključe ignoriramo, saj nas ne zanimajo. Kot izhod iz preslikovanja postavimo **leto** kot ključ in **temperaturo** kot vrednost.

KLJUČ, VREDNOST
(2013, 15)
(2012, -5)
(2012, 3)
(2013, -2)
(2013,23)

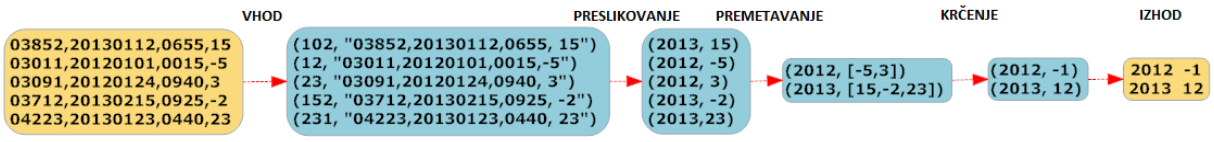
Preden MapReduce pošlje zgornji nabor vrstic v krčenje, se zgodi še vmesni korak. Ta korak imenujemo premetavanje (*angl. shuffle*) in služi sortiranju in združevanju vrstic po ključu. Vhod v funkcijo krčenja je po premetavanju takšen (premetavanje je uredilo ključ po naraščajočem vrstnem redu):

KLJUČ, NABOR VREDNOSTI ZA KLJUČ
(2012, [-5,3])
(2013, [15,-2,23])

Vsako leto je sedaj v svoji vrstici, dodan pa mu je nabor vrednosti za leto. Vse, kar mora sedaj funkcija krčenja narediti, je, da za vsako leto iz pripravljenih naborov izračuna povprečno temperaturo. Tako dobimo končni izhod:

KLJUČ, POVPREČNA TEMPERATURA
(2012, -1)
(2013, 12)

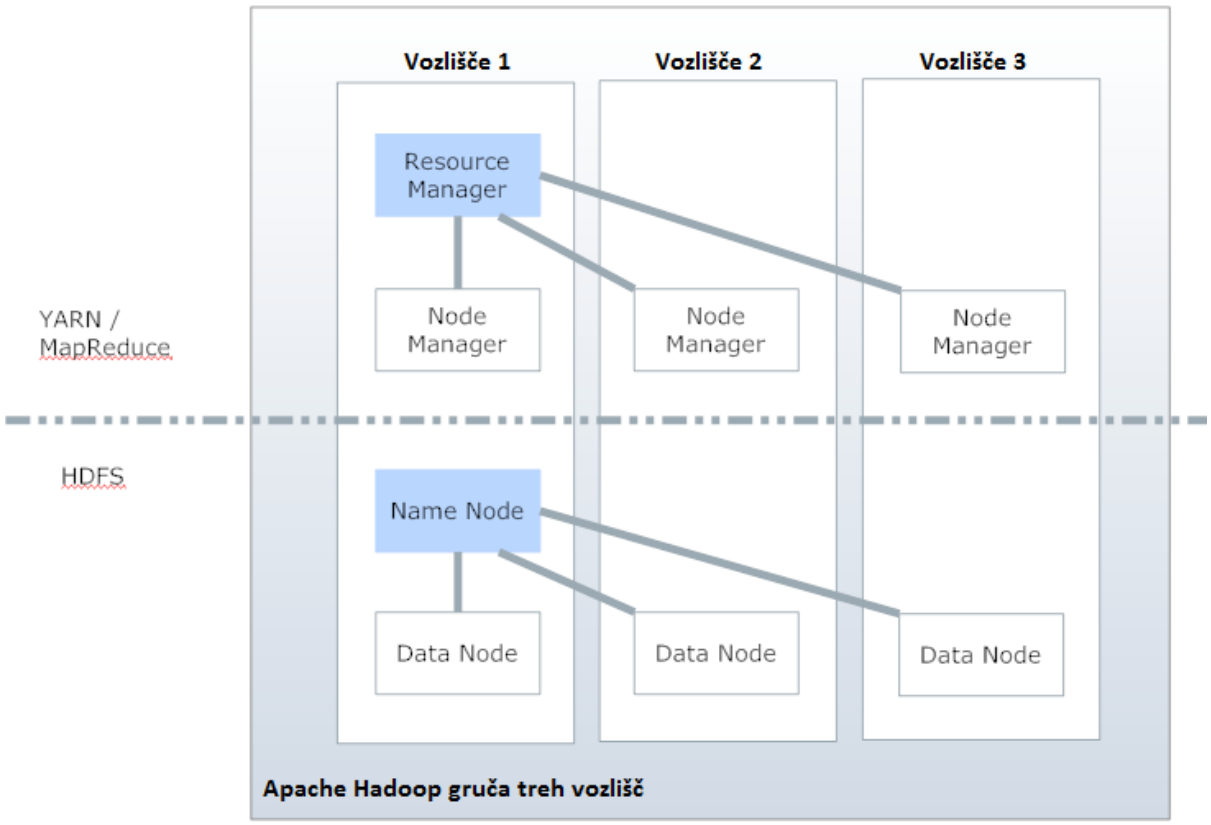
Slika 6 je grafični prikaz poteka zgornje MapReduce obdelave.



Slika 6: Prikaz delovanja osnovnih korakov MapReduce [1]

2.3 Videz tipične gruče Apache Hadoop

Da stvari iz poglavja 2.1 in 2.2 povežemo v celoto, sledi grafičen opis tipične gruče Apache Hadoop. Osnovna namestitev Hadoop vsebuje HDFS, YARN in MapReduce. Če imamo v gruči tri računalnike, bi bila postavitev videti takole (glej sliko 7):



Slika 7: Prikaz tipične arhitekture Apache Hadoop na primeru treh vozlišč. Slika ponazarja lokalno arhitekturo uporabljeno v četrtem poglavju [5]

Poglavje 3 Implementacija

Izmišljeno podjetje, ki razpolaga z vremenskimi postajami, potrebuje sistem, ki bi mu omogočal enostavno analizo podatkov o vremenu. Proti plačilu želi zagotavljati dostop do podatkov meteorološkim inštitutom. Podjetje ima v arhivu za sto let zgodovinskih podatkov v tekstovni obliki. Načrtujejo občutno povečanje števila postaj, kar bi pomenilo povečanje v količini podatkov. Podjetje uporablja rešitev v standardni relacijski podatkovni shrambi, ki zaradi zagotavljanja sprejemljive hitrosti poizvedb ne podpira poizvedovanja po celotnem obsegu arhiva. Prav tako so starejši podatki drugače strukturirani in vsebujejo manj informacij. Podjetje najame podizvajalce, ki opravijo analizo zahtev. Za shranjevanje podatkov je predlagan HDFS, kot ogrodje za poganjanje poizvedb pa MapReduce oz. Hadoop podprojekti, ki uporabljajo programski model MapReduce za poizvedovanje (npr. Pig). S podizvajalci se dogovorijo za pilotski projekt, ki bo pokazal zmogljivosti sistema. Ker nimajo na voljo moderne strežniške arhitekture, se za pilotski projekt odločijo uporabiti Amazon EMR in S3 v oblaku, zavedajoč se, da bodo obdelave trajale nekoliko dlje, saj podatki za pilota ne bodo shranjeni v HDFS.

Cilj pilota je implementirati reprezentativno aplikacijo, ki bo iz vremenskih podatkov na urnem nivoju izračunala povprečno temperaturo po letih in postajah ter podatke združila z lokacijo in imenom vremenskih postaj. Aplikacija bo omogočala poganjanje na zahtevo v oblaku ali lokalnem okolju.

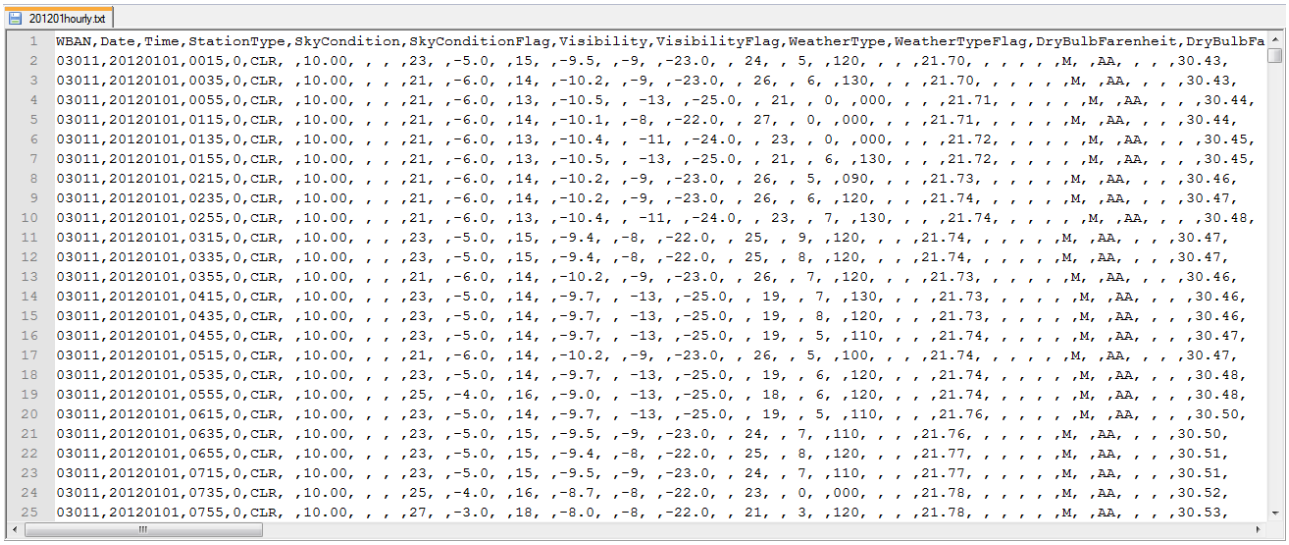
3.1 Struktura podatkov

Kot vir sem izbral vremenske podatke iz meteoroloških postaj v Združenih državah Amerike [13]. Podatki so v tekstovni obliki in razdeljeni na več datotek. Na voljo imam podatke zadnjih šest let. Skupna velikost vseh datotek je 35 GB (*angl. gigabyte*). Na voljo imam dva tipa datotek, in sicer datoteke z vremenskimi meritvami postaj in datoteke, ki vsebujejo podatke o postajah. Vremenska datoteka je poimenovana s frekvenco hranjenja podatkov, letom in mesecem (npr. hourly201201.txt). Datoteke s podatki postaj so poimenovane z drugim delom imena vremenske datoteke, na koncu pa imajo oznako, da gre za postaje (npr. 201201station.txt). Vsaka datoteka ima na začetku vrstico z glavo, ki opisuje podatke. Podatki so v vremenski datoteki ločeni z vejico, v datoteki postaj pa s poševnico. Vrstice se končajo z

znakom za novo vrstico. Številke uporabljajo piko kot ločilo za decimalke. Tabeli 3 in 4 prikazujeta stolpce uporabljene pri implementaciji, sliki 8 in 9 pa reprezentativni set celotne strukture podatkov o vremenu in postajah.

Številka stolpca	Šifra	Opis
1	WBAN	Šifra vremenske postaje
2	Date	Datum
13	DryBulbCelsius	Vrednost suhega termometra v stopinjah Celzija

Tabela 3: Prikaz stolpcev, ki bodo uporabljeni v obdelavi podatkov o vremenu



201201hourly.txt

```
1  WBAN,Date,Time,StationType,SkyCondition,SkyConditionFlag,Visibility,VisibilityFlag,WeatherType,WeatherTypeFlag,DryBulbFarenheit,DryBulbFa
2  03011,20120101,0015,0,CLR, ,10.00, , , ,23, ,-5.0, ,15, ,-9.5, ,-9, ,-23.0, , 24, , 5, ,120, , , ,21.70, , , , , ,M, ,AA, , , ,30.43,
3  03011,20120101,0035,0,CLR, ,10.00, , , ,21, ,-6.0, ,14, ,-10.2, ,-9, ,-23.0, , 26, , 6, ,130, , , ,21.70, , , , , ,M, ,AA, , , ,30.43,
4  03011,20120101,0055,0,CLR, ,10.00, , , ,21, ,-6.0, ,13, ,-10.5, , -13, ,-25.0, , 21, , 0, ,000, , , ,21.71, , , , , ,M, ,AA, , , ,30.44,
5  03011,20120101,0115,0,CLR, ,10.00, , , ,21, ,-6.0, ,14, ,-10.1, ,-8, ,-22.0, , 27, , 0, ,000, , , ,21.71, , , , , ,M, ,AA, , , ,30.44,
6  03011,20120101,0135,0,CLR, ,10.00, , , ,21, ,-6.0, ,13, ,-10.4, , -11, ,-24.0, , 23, , 0, ,000, , , ,21.72, , , , , ,M, ,AA, , , ,30.45,
7  03011,20120101,0155,0,CLR, ,10.00, , , ,21, ,-6.0, ,13, ,-10.5, , -13, ,-25.0, , 21, , 6, ,130, , , ,21.72, , , , , ,M, ,AA, , , ,30.45,
8  03011,20120101,0215,0,CLR, ,10.00, , , ,21, ,-6.0, ,14, ,-10.2, ,-9, ,-23.0, , 26, , 5, ,090, , , ,21.73, , , , , ,M, ,AA, , , ,30.46,
9  03011,20120101,0235,0,CLR, ,10.00, , , ,21, ,-6.0, ,14, ,-10.2, ,-9, ,-23.0, , 26, , 6, ,120, , , ,21.74, , , , , ,M, ,AA, , , ,30.47,
10 03011,20120101,0255,0,CLR, ,10.00, , , ,21, ,-6.0, ,13, ,-10.4, , -11, ,-24.0, , 23, , 7, ,130, , , ,21.74, , , , , ,M, ,AA, , , ,30.48,
11 03011,20120101,0315,0,CLR, ,10.00, , , ,23, ,-5.0, ,15, ,-9.4, ,-8, ,-22.0, , 25, , 9, ,120, , , ,21.74, , , , , ,M, ,AA, , , ,30.47,
12 03011,20120101,0335,0,CLR, ,10.00, , , ,23, ,-5.0, ,15, ,-9.4, ,-8, ,-22.0, , 25, , 8, ,120, , , ,21.74, , , , , ,M, ,AA, , , ,30.47,
13 03011,20120101,0355,0,CLR, ,10.00, , , ,21, ,-6.0, ,14, ,-10.2, ,-9, ,-23.0, , 26, , 7, ,120, , , ,21.73, , , , , ,M, ,AA, , , ,30.46,
14 03011,20120101,0415,0,CLR, ,10.00, , , ,23, ,-5.0, ,14, ,-9.7, , -13, ,-25.0, , 19, , 7, ,130, , , ,21.73, , , , , ,M, ,AA, , , ,30.46,
15 03011,20120101,0435,0,CLR, ,10.00, , , ,23, ,-5.0, ,14, ,-9.7, , -13, ,-25.0, , 19, , 8, ,120, , , ,21.73, , , , , ,M, ,AA, , , ,30.46,
16 03011,20120101,0455,0,CLR, ,10.00, , , ,23, ,-5.0, ,14, ,-9.7, , -13, ,-25.0, , 19, , 5, ,110, , , ,21.74, , , , , ,M, ,AA, , , ,30.47,
17 03011,20120101,0515,0,CLR, ,10.00, , , ,21, ,-6.0, ,14, ,-10.2, ,-9, ,-23.0, , 26, , 5, ,100, , , ,21.74, , , , , ,M, ,AA, , , ,30.47,
18 03011,20120101,0535,0,CLR, ,10.00, , , ,23, ,-5.0, ,14, ,-9.7, , -13, ,-25.0, , 19, , 6, ,120, , , ,21.74, , , , , ,M, ,AA, , , ,30.48,
19 03011,20120101,0555,0,CLR, ,10.00, , , ,25, ,-4.0, ,16, ,-9.0, , -13, ,-25.0, , 18, , 6, ,120, , , ,21.74, , , , , ,M, ,AA, , , ,30.48,
20 03011,20120101,0615,0,CLR, ,10.00, , , ,23, ,-5.0, ,14, ,-9.7, , -13, ,-25.0, , 19, , 5, ,110, , , ,21.76, , , , , ,M, ,AA, , , ,30.50,
21 03011,20120101,0635,0,CLR, ,10.00, , , ,23, ,-5.0, ,15, ,-9.5, ,-9, ,-23.0, , 24, , 7, ,110, , , ,21.76, , , , , ,M, ,AA, , , ,30.50,
22 03011,20120101,0655,0,CLR, ,10.00, , , ,23, ,-5.0, ,15, ,-9.4, ,-8, ,-22.0, , 25, , 8, ,120, , , ,21.77, , , , , ,M, ,AA, , , ,30.51,
23 03011,20120101,0715,0,CLR, ,10.00, , , ,23, ,-5.0, ,15, ,-9.5, ,-9, ,-23.0, , 24, , 7, ,110, , , ,21.77, , , , , ,M, ,AA, , , ,30.51,
24 03011,20120101,0735,0,CLR, ,10.00, , , ,25, ,-4.0, ,16, ,-8.7, ,-8, ,-22.0, , 23, , 0, ,000, , , ,21.78, , , , , ,M, ,AA, , , ,30.52,
25 03011,20120101,0755,0,CLR, ,10.00, , , ,27, ,-3.0, ,18, ,-8.0, ,-8, ,-22.0, , 21, , 3, ,120, , , ,21.78, , , , , ,M, ,AA, , , ,30.53,
```

Slika 8: Vsebina datoteke z vremenskimi podatki

Številka stolpca	Šifra	Opis
1	WBAN	Šifra vremenske postaje
9	Location	Opisna lokacija
10	Latitude	Zemljepisna širina
11	Longitude	Zemljepisna dolžina

Tabela 4: Prikaz stolpcev, ki bodo uporabljeni v obdelavi podatkov o postajah

201201station.txt

```
WBAN|WMO|CallSign|ClimateDivisionCode|ClimateDivisionStateCode|ClimateDivisionStationCode|Name|State|I
03011||TEX||||TELLURIDE|CO|TELLURIDE REGIONAL AIRPORT|37.954|-107.901|9078|9078||-7
03012||SKX||||TAOS|NM|TAOS REGIONAL AIRPORT|36.458|-105.667|7091|7091||-7
03013||LAA||||LAMAR|CO|LAMAR MUNICIPAL AIRPORT|38.070|-102.688|3683|3703|3675|-7
03014||4SL||||TORREON|NM|TORREON|35.789|-107.248|0|6909|0|-7
03016||RIL||||RIFLE|CO|GARFIELD CO REGIONAL ARPT|39.526|-107.726|5498|5544|5506|-7
03017|72565|DEN|04|05|2220|DENVER|CO|DENVER INTERNATIONAL AIRPORT|39.833|-104.658|5437|5431|5382|-7
03024||BGD||||BORGER|TX|HUTCHINSON COUNTY AIRPORT|35.700|-101.394|3040|3054|3041|-6
03026||ITR||||BURLINGTON|CO|KIT CARSON COUNTY AIRPORT|39.245|-102.284|4192|4217|4198|-7
03027||CQC||||CLINES CORNERS|NM|CLINES CORNERS|35.003|-105.663|7086|7086|7092|-7
03028||SPD||||SPRINGFIELD|CO|COMANCHE NATIONAL GRASSLAND|37.283|-102.614|4880|4380|4386|-7
03029||RQE||||WINDOW ROCK|AZ|WINDOW ROCK AIRPORT|35.658|-109.061|6733|6739|6745|-7
03030||GUY||||GUYMON|OK|GUYMON MUNICIPAL AIRPORT|36.682|-101.505|3112|3112|3118|-6
03031||ODO||||ODESSA|TX|ODESSA-SCHLEMEYER FLD ARPT|31.921|-102.387|2964|3001|2962|-6
03032||6R6||||DRYDEN|TX|TERRELL COUNTY AIRPORT|30.048|-102.213|274|2301|2307|-6
03033||0E0||||MORIARTY|NM|MORIARTY AIRPORT|34.985|-106.000|6199|6199||-7
03034||AEG||||ALBUQUERQUE|NM|DOUBLE EAGLE II AIRPORT|35.145|-106.784|5837|5837||-7
03035||ATS||||ARTESIA|NM|ARTESIA MUNICIPAL AIRPORT|32.853|-104.467|3541|3541||-7
03036||BDG||||BLANDING|UT|BLANDING MUNICIPAL AIRPORT|37.583|-109.583|5865|5865||-7
03037||CBK||||COLBY|KS|SHALTZ FIELD AIRPORT|39.428|-101.034|3186|3186||-6
03038||CCU||||COPPER MOUNTAIN|CO|COPPER MOUNTAIN|39.467|-106.150|0|12074|0|-7
03039||CPW||||WOLF CREEK PASS|CO|WOLF CREEK PASS AWOS-3 AIRPORT|37.450|-106.800|11791|11791||-7
03040||MNH||||MONUMENT HILL|CO|MONUMENT HILL AWOS-3 ARPT|39.217|-104.633|7060|7060||-7
03041||MYP||||MONARCH PASS|CO|MONARCH PASS AWOS-3 ARPT|38.483|-106.317|12031|12031||-7
03042||VTP||||LA VETA PASS|CO|LA VETA PASS AWOS-3 ARPT|37.500|-105.167|10217|10217||-7
```

Slika 9: Vsebina datoteke s podatki o postajah

3.2 Razvoj aplikacije Java BMapReduce

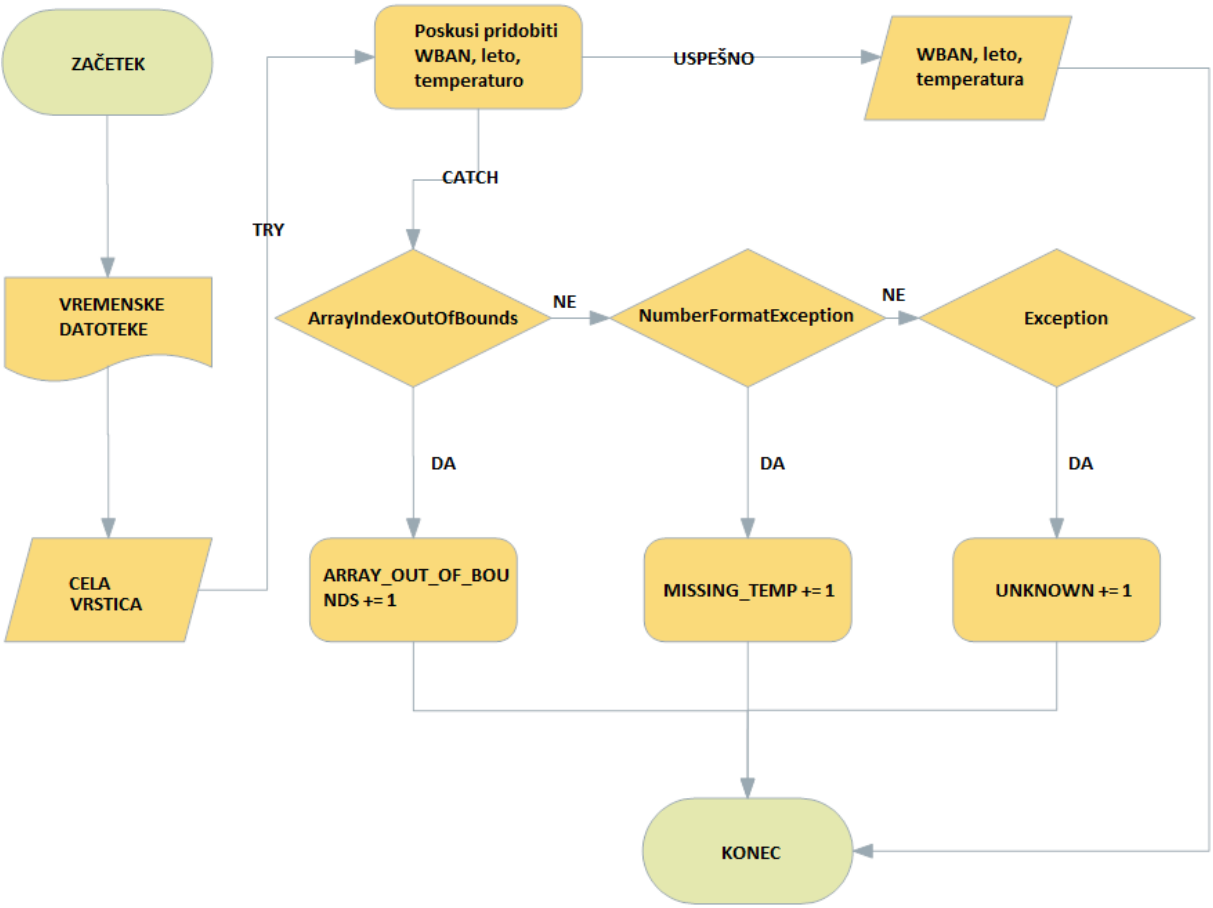
Za razvoj aplikacije Java sem izbral razvojno okolje Eclipse. Aplikacija vsebuje šest razredov:

- BMapReduce.java
- BMapper.java
- BReducer.java
- BCombiner.java
- Station.java
- StationMap.java
- BMapReduceTest.java

BMapReduce vsebuje glavno metodo in skrbi za inicializacijo obdelave MapReduce. V glavni metodi se poda razrede, ki bodo uporabljeni v posameznih korakih MapReduce. Potrebno je določiti, kateri razredi bodo uporabljeni za krčenje, preslikovanje in združevanje. Poleg teh definicij je potrebno podati ime obdelave, tip izhodnega ključa, tip izhodne vrednost in format

vhodnih podatkov. Kot vhod v obdelavo MapReduce sem dodal še datoteko z informacijami o postajah, ki jih v koraku krčenja združim s povprečno temperaturo preko postaje WBAN.

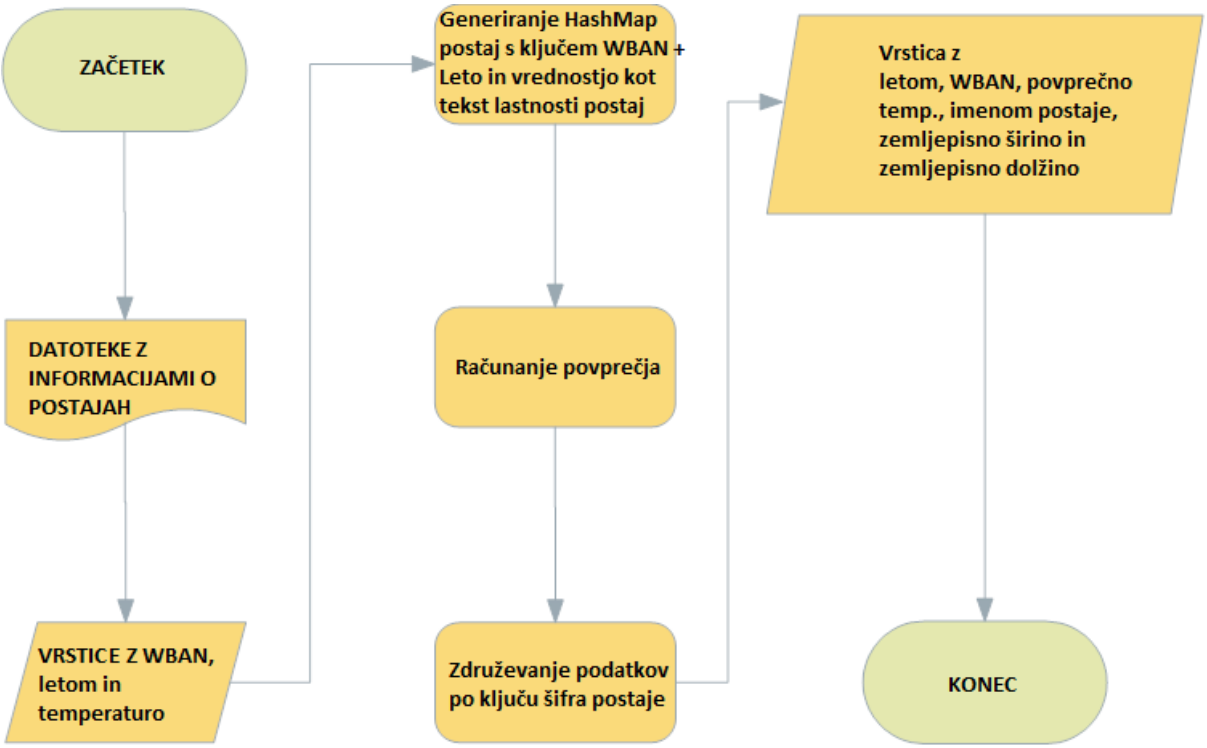
Razred BMapper vsebuje programsko logiko MapReduce preslikovanja. Skrbi za transformacijo vhodnih podatkov. Vhod v BMapper so vrstice posameznih datotek. Programer lahko izbira med različnimi tipi vhoda ali implementira svojega. Iz vrstice vzamem informacije o temperaturi, datumu in šifri vremenske postaje, ki služijo kot izhodni nabor podatkov za nadaljnjo obdelavo. V tem razredu je implementirana logika za iskanje napačno oblikovanih vrstic. Če pride do napačnih vrstic oz. vrstice ne vsebujejo podatka o temperaturi, se informacija o tem izpiše v števce napak. Števci napak so prikazani na koncu obdelave v informacijo uporabniku. Slika 10 prikazuje delovanje razreda BMapper.



Slika 10: Diagram poteka razreda BMapper

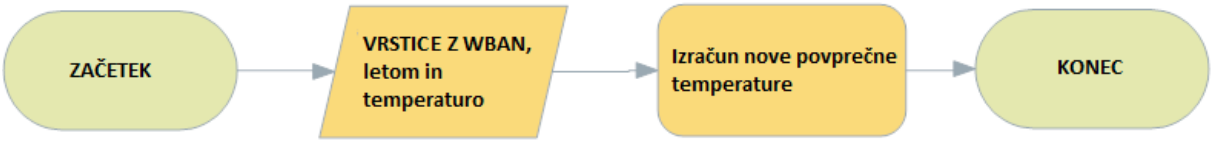
BReducer služi kot implementacija krčenja MapReduce. Vhod v BReducer je datoteka z informacijami o postajah podana v glavni metodi BMapReduce ter izhodna vrstica iz razreda BMapper. BReducer poskrbi za izračun povprečja po postajah in letih. Prav tako BReducer

opravi združevanje podatkov o postajah s podatki o povprečnih temperaturah po letih za postajo. Ključ za združevanje in račun povprečja je šifra postaje. Slika 11 prikazuje delovanje razreda BReducer.



Slika 11: Diagram poteka razreda BReducer

Razred BCombiner je implementacija združevanja MapReduce. Služi kot pomoč izračunu povprečnih temperatur in pripomore k hitrejšemu izračunu manjših delov celotnega obsega podatkov. Če združevanje ni definirano, potem MapReduce za združevanje uporabi krčenje. V moji implementaciji združevanje opravlja nalogo računanja povprečja vmesnih korakov obdelave. Slika 12 prikazuje delovanje razreda BCombiner.



Slika 12: Diagram poteka razreda BCombiner

Razred Station opisuje objekt postaje in vsebuje metode za nastavljanje lastnosti:

- setWban(String)/getWban()
- setLocation(String)/getLocation()
- setLatitude(String)/getLatitude()
- setLongitude(String)/getLongitude()
- setup(String vrstica)

Razred se inicializira preko metode *setup(String)* poklicane v konstruktorju razreda, ki sprejme vrstico s podatki ločeni z vejico in jih zapiše v pripadajoče lastnosti razreda. Razred StationMap skrbi za inicializacijo Java HashMap objekta iz datoteke postaj za potrebe združevanja podatkov o vremenu po šifri postaje. Delovanje razreda BMapReduceTest je opisano v poglavju 3.4 kjer opisujem testiranje z MRUnit. Programska koda implementiranih Java razredov je na voljo v dodatku A.

3.3 Razvoj skripte Pig Latin za pridobivanje podatkov o vremenskih postajah

Pig Latin je skriptni jezik, ki omogoča poizvedbe po veliki količini podatkov. Kritike MapReduce so predvsem povezane s kompleksnostjo. Razvojni cikel kompleksnejših obdelav, kot so npr. združevanje podatkov (*angl. join*), je lahko zelo dolg. Poleg tega je ponekod potrebno razviti več MapReduce aplikacij za eno obdelavo. Zaradi tega so pri Yahoo! razvili Pig. Pig služi kot interpreter iz skripte Pig Latin v MapReduce obdelave. V ozadju kliče vnaprej programirane MapReduce procedure. S pomočjo Pig so lahko raziskovalci pri Yahoo! enostavno in hitro sami napisali želene poizvedbe. Pig lahko obdela več terabajtov podatkov v le nekaj vrsticah skripte. Pig lahko namestimo v lokalnem načinu v testne namene in načinu HDFS za uporabo Hadoop. Do Pig okolja dostopamo preko konzolnega vmesnika Grunt, ki nam omogoča poganjanje Pig Latin ukazov in izpis rezultatov poizvedb. Pig vmesnih korakov poizvedbe ne shranjuje na disk, ampak nam prikazuje rezultate nad manjšim delom podatkov. Šele takrat, ko shranimo rezultate Pig obdelav z ukazom *store,* se požene obdelava MapReduce in rezultati se zapišejo v izhodno datoteko [1,4].

Pig omogoča sprotno pregledovanje rezultatov z ukazom *illustrate*. S Pig bom predstavil način razvoja obdelav brez pisanja MapReduce javanske kode. Pig Latin skripta bo kot

rezultat iz vseh datotek, ki vsebujejo lastnosti postaj, izluščila neponavljajoče postaje glede na WBAN. Če je za WBAN podano več različnih postaj, vzamemo prvo, ki se pojavi. Z ukazom *illustrate* bom prikazal korake Pig Latin skripte.

```
grunt> postaje = LOAD 'hdfs://hsingle:10001/data/stations/*'
grunt> USING PigStorage('|')
grunt> AS
grunt>(WBAN,WMO,CallSign,ClimateDivisionCode,ClimateDivisionStateCode,ClimateDivi
sionStationCode,Name,State,Location,Latitude,Longitude,GroundHeight,StationHeight,Baro
meter,TimeZone);
grunt> postaje_sifra_ime_lokacija = FOREACH postaje GENERATE WBAN, Location,
Latitude, Longitude;
grunt> illustrate postaje_sifra_ime_lokacija;
-------------------------------------------------------------------------------------------------
postaje_sifra_ime_lokacija
-------------------------------------------------------------------------------------------------
| WBAN          | Location                                | Latitude   | Longitude   |
-------------------------------------------------------------------------------------------------
| 94830         | TOLEDO EXPRESS AIRPORT          | 41.589     | -83.801       |
-------------------------------------------------------------------------------------------------
grunt> postaje_po_WBAN = GROUP postaje_sifra_ime_lokacija BY WBAN;
grunt> illustrate postaje_po_WBAN;
-------------------------------------------------------------------------------------------------
postaje_sifra_ime_lokacija
-------------------------------------------------------------------------------------------------
| WBAN          | Location                                | Latitude   | Longitude   |
-------------------------------------------------------------------------------------------------
| 94830         | TOLEDO EXPRESS AIRPORT          | 41.589     | -83.801       |
| 94830         | TOLEDO EXPRESS AIRPORT          | 41.589     | -83.801       |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
postaje_po_WBAN group:bytearray
-------------------------------------------------------------------------------------------------------------
| WBAN        | postaje_sifra_ime_lokacija:bag{:tuple(WBAN,Location,Latitude,Longitude)}
-------------------------------------------------------------------------------------------------------------
```

```
| 94830         |{(94830, TOLEDO EXPRESS AIRPORT,-83.801),(94830,TOLE..,-83.801)}
-------------------------------------------------------------------------------------------------------------------
grunt> prva_postaja_v_mapi = FOREACH postaje_po_WBAN {
grunt> prva_postaja = TOP(1,1,postaje_sifra_ime_lokacija);
grunt> GENERATE flatten(prva_postaja); }
grunt> illustrate prva_postaja_v_mapi;
-------------------------------------------------------------------------------------------------------------
postaje_sifra_ime_lokacija
-------------------------------------------------------------------------------------------------------------
| WBAN          | Location                                  | Latitude   | Longitude   |
-------------------------------------------------------------------------------------------------------------
| 94830         | TOLEDO EXPRESS AIRPORT          | 41.589     | -83.801       |
| 94830         | TOLEDO EXPRESS AIRPORT          | 41.589     | -83.801       |
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
postaje_po_WBAN group:bytearray
-------------------------------------------------------------------------------------------------------------------
| WBAN         | postaje_sifra_ime_lokacija:bag{:tuple(WBAN,Location,Latitude,Longitude)}
-------------------------------------------------------------------------------------------------------------------
| 94830         |{(94830, TOLEDO EXPRESS AIRPORT,-83.801),(94830,TOLE..,-83.801)}
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
prva_postaja_v_mapi
-------------------------------------------------------------------------------------------------------------
| WBAN          | Location                                  | Latitude   | Longitude   |
-------------------------------------------------------------------------------------------------------------
| 94830         | TOLEDO EXPRESS AIRPORT          | 41.589     | -83.801       |
-------------------------------------------------------------------------------------------------------------
grunt> rezultat_postaje = FOREACH prva_postaja_v_mapi
GENERATE flatten(WBAN),flatten(Location), flatten(Latitude),flatten(Longitude);
grunt> STORE rezultat_postaje INTO
'hdfs://hsingle:10001/data/output/stations_unique.txt' USING PigStorage(',');
```

3.4 Testiranje in MRUnit

Podatki, ki jih želimo obdelati, so lahko neurejeni in vsebujejo napake, ki jih zaradi velike količine podatkov težko zaznamo vnaprej. Če imamo v obdelavi več sto datotek, ki lahko

prihajajo iz različnih virov in so različno urejene, je nemogoče vnaprej predvideti vse težave in nepravilnosti v podatkih. Pred poganjanjem v oblaku ali v lokalnem okolju je dobro obdelave, ki lahko trajajo dlje časa, temeljito testirati. Z nepravilnimi rezultati izgubljamo čas, denar in razpoložljivost računalniških virov. S testiranjem želimo zmanjšati število pognanih obdelav do želenega rezultata, kar je ključno za poganjanje v oblaku in lokalni gruči v produkcijskem okolju. Obdelave MapReduce lahko poganjamo tudi lokalno brez HDFS in YARN. Enako velja za Pig. Lokalno poganjanje obdelav nad manjšim setom reprezentativnih podatkov je stalna praksa pri testiranju aplikacij za obdelavo velikih količin podatkov.

MRUnit je odprtokodna zbirka Java JUnit testnih knjižnic namenjenih za testiranje obdelav MapReduce. Pri testiranju MapReduce programov je potrebno preveriti pravilnost korakov obdelave (preslikovanje, krčenje, premetavanje). Za testiranje sem razvil šest metod, štiri za preslikovanje in eno za krčenje.

- basicMapperInputTest,
- basicMapperInputTestArrayIndexOutOfBounds,
- basicMapperInputTestHeader,
- basicMapperInputMissing,
- basicReducerInputTest

Metoda basicMapperInputTest služi preverjanju pravilnosti obdelave enostavne vrstice. Vhod je vrstica v osnovni pričakovani obliki. Pričakovana izhoda sta leto v obliki 2010 in vrednost temperature npr. -7.0. Metoda basicMapperInputTestArrayIndexOutOfBounds preverja obnašanje preslikovanja, ko pride do napak v vrstici in program ne najde temperature na pričakovanem mestu. Pričakovan izhod je povečan števec ARRAY_OUT_OF_BOUNDS. BasicMapperInputTestHeader služi preverjanju izvedbe preslikovanja, ko program naleti na vnos vrstice z glavo. Pričakovan vhod je vrstica z glavo, pričakovanega izhoda ni, saj se ta vrstica ignorira. Metoda basicMapperInputMissing preverja pravilnost preslikovanja, ko program pride do vrstice, iz katere ni mogoče izluščiti temperature. Podatek je lahko napačnega numeričnega formata ali podatka sploh ni na pričakovanem mestu. Vhod je vrstica, kjer podatek manjka, izhod pa povečan števec MISSING_TEMP. BasicReducerInputTest preverja izračun povprečne temperature in povezovanje lastnosti postaj z izračunanimi temperaturami po letih za WBAN. Vhod je datoteka s podatki o postajah ter vrstica iz datoteke s temperaturami po postajah. Pričakovan izhod pa izračunana povprečna temperatura po letih in šifri postaje združena s podatki o postaji.

3.5 Lokalna infrastruktura

Lokalno infrastrukturo predstavljajo trije prenosniki. V lokalno omrežje so povezani preko usmerjevalnika, ki ima tudi funkcijo brezžične povezave. Na vsakem je nameščen Oracle VirtualBox, v katerem je pognan virtualni računalnik z Ubuntu Linux 12.04 LTS (*angl. Long Term Support*).

Specifikacije strojne opreme:

- HP EliteBook 8560w (8GB RAM, 8CPU i7 – za virtualizacijo 5GB RAM, 4CPU, disk s 7200 obratov na minuto, 3072MB namenjenih za YARN, povezan preko kabla) – hsingle
- Acer Aspire 5741g (4GB RAM, 4CPU i3 – za virtualizacijo 2GB RAM, 2CPU, disk s 5400 obratov na minuto, 1536MB namenjenih za YARN, povezan preko kabla) – hsingle2
- Sony VAIO VPCEB4L1E (4GB ram, 2 CPU i3 – za virtualizacijo 2GB RAM, 1 CPU, disk s 5400 obratov na minuto, 1024MB namenjenih za YARN, povezan preko brezžičnega omrežja) – hsingle3
- brezžični usmerjevalnik (prepustnost 100 Mbit/s preko kabla in 54Mbit/s preko brezžičnega omrežja). Hsingle in hsingle2 sta povezana preko kabla, hsingle3 pa preko brezžičnega omrežja saj usmerjevalnik, ki sem ga imel na voljo podpira samo dva računalnika povezana preko kabla v internem omrežju.

Vsak računalnik ima statičen IP (*angl. Internet Protocol*) naslov, nastavljeno ime ime v /etc/hostname in bližnjice imen drugih računalnikov gruče v /etc/hosts.

Iz spletne strani Apache sem prenesel Hadoop verzijo 2.2.0 za Linux 64bit. Po prenosu sem datoteko razširil v /usr/local/hadoop. S strani sem prenesel še Pig in ga razširil v /usr/local/pig.

Namestitev Hadoop vključuje nastavljanje parametrov XML datotek. Nastavljanje preko XML datotek omogoča veliko mero avtomatizma in prav zaradi tega je Hadoop gručo enostavno postaviti tako v lokalnem okolju kot v oblaku, saj se vsako odvisno vozlišče enostavno kopira. Edina razlika je ime vozlišča in IP naslov. Vsaka komponenta Hadoop ima svoje datoteke za nastavitve:

- HDFS - hdfs-site.xml, core-site.xml,

- MapReduce – mapred-site.xml
- YARN – yarn-site.xml

Vsebina nastavitvenih datotek je prikazana v dodatku B za vsako vozlišče posebej. Prikazane so najosnovnejše nastavitve za delovanje gruče. Obstaja veliko mikro nastavitev, s katerimi lahko v detajle nastavimo gručo. Te so dostopne preko uradne spletne strani Apache http://hadoop.apache.org/docs/r2.2.0/hadoop-project-dist/hadoop-common/ClusterSetup.html.

Poleg posameznih datotek za namestitev obstajajo še .sh datoteke, ki nam pomagajo nastaviti spremenljivke za okolje (JAVA_HOME, YARN_HOME, HDFS_HOME itd.) in datoteke z informacijo o vozliščih gruče (master, slave datoteki). V .bashrc uporabnika, ki bo zaganjal procese Hadoop, je priporočljivo nastaviti še naslednje globalne spremenljivke:

```
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_LOG_DIR=$HADOOP_COMMON_HOME/logs
export HADOOP_HDFS_HOME=$HADOOP_COMMON_HOME
export HADOOP_CONF_DIR=$HADOOP_COMMON_HOME/etc/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_COMMON_HOME
export HADOOP_YARN_HOME=$HADOOP_COMMON_HOME
export MAPRED_CONF_DIR=$HADOOP_CONF_DIR
export YARN_CONF_DIR=$HADOOP_CONF_DIR
export
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_COMMON_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_COMMON_HOME/lib"
export PIG_COMMON_HOME=/usr/local/pig
export PIG_CONF_DIR=$PIG_COMMON_HOME/config
export PIG_CLASSPATH=$PIG_COMMON_HOME/pig-
0.12.0.jar:$HADOOP_CONF_DIR
export
PATH=$PATH:$HADOOP_COMMON_HOME/bin:$HADOOP_COMMON_HOME/sbin:$
PIG_COMMON_HOME/bin
```

Po konfiguraciji XML datotek je gruča Hadoop pripravljena za uporabo. Preden poženemo procese HDFS in YARN, moramo pred prvo uporabo formatirati glavno vozlišče. To storimo z ukazom:

```
hsingle@hsingle:~$ hdfs namenode –format
hsingle has been successfully formated.
```

Sedaj samo še poženemo HDFS in YARN, in gruča Hadoop je pripravljena za uporabo.

```
hsingle@hsingle:~$ start-dfs.sh
hsingle: starting namenode
hsingle: starting datanode
hsingle3: starting datanode
hsingle2: starting datanode
Hadoop started sucessfully
hsingle@hsingle:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager
hsingle: starting nodemanager
hsingle3: starting nodemanager
hsingle2: starting nodemanager
YARN started sucessfully
```

Poglavje 4 Poganjanje obdelav

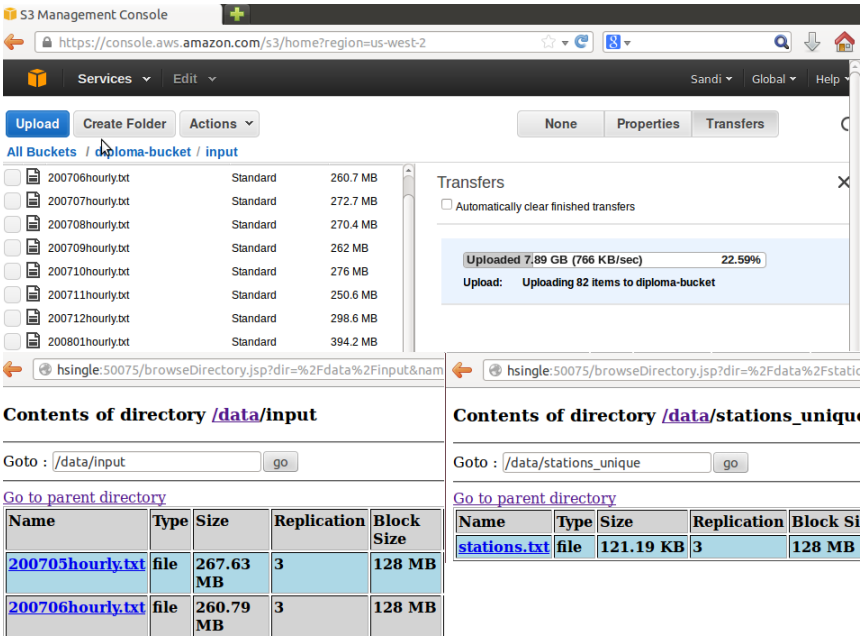
V prejšnjem poglavju smo videli potek implementacije MapReduce javanske aplikacije in skripte Pig Latin. Sedaj je z obdelavo MapReduce potrebno združiti podatke o vremenu s podatki o vremenskih postajah. Preden začnemo s poganjanjem, je v HDFS potrebno prenesti podatke o postajah in izhod iz Pig Latin skripte. Najprej naredimo mape za podatke, potem pa izvedemo ukaze za prenos.

```
hsingle@hsingle:~$ hdfs dfs –mkdir /data
hsingle@hsingle:~$ hdfs dfs –mkdir /data/input
hsingle@hsingle:~$ hdfs dfs –mkdir /output
hsingle@hsingle:~$ hdfs dfs –mkdir /stations_unique
```

Sedaj imamo pripravljene mape in lahko poženemo prenos podatkov v HDFS.

```
hsingle@hsingle:~$ hdfs dfs –copyFromLocal /home/hsingle/podatki/postaje/*.txt
hdfs://hsingle:10001/data/input/
hsingle@hsingle:~$ hdfs dfs –copyFromLocal /home/hsingle/podatki/PIG/stations.txt
hdfs://hsingle:10001/data/input/postaje/stations.txt
```

Za potrebe obdelav v oblaku moramo podatke naložiti v S3. Ta korak je enostaven in poteka v celoti skozi spletni uporabniški vmesnik (glej sliko 13).



Slika 13: Prikaz uvoženih podatkov v S3 in lokalni HDFS

Hraniti podatke v oblaku S3 je cenovno ugodna rešitev. Na mesec je potrebno plačati približno 0,02 EUR/GB za podatke pod 1TB. Poleg tega se prenos podatkov v oblak ne zaračunava. [2]

4.1 Poganjanje BMapReduce v lokalnem okolju

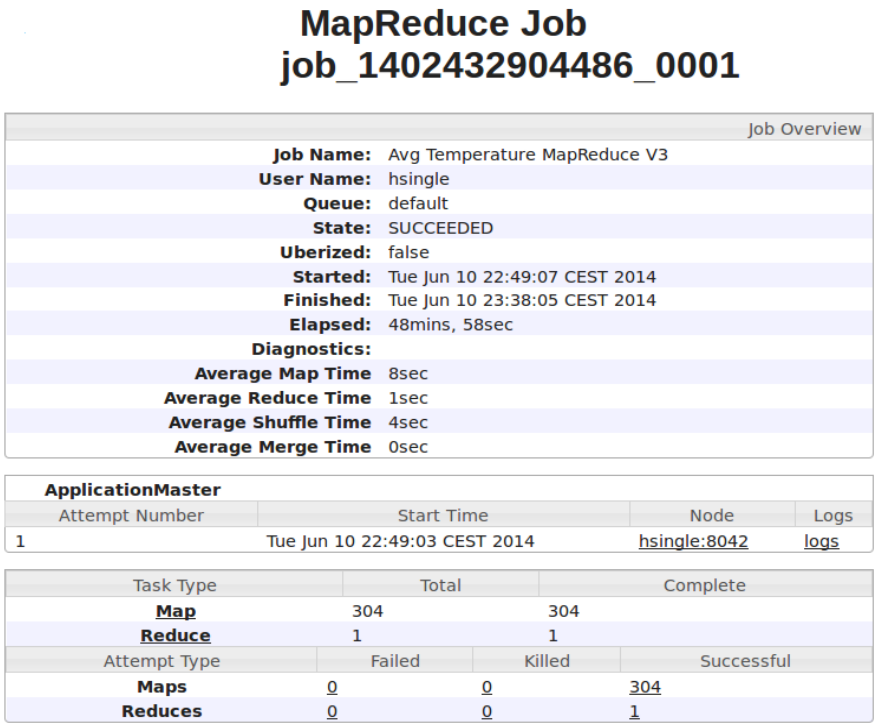
V poglavju 3.2 smo implementirali javansko aplikacijo BMapReduce. Aplikacijo moramo pognati in preveriti rezultate obdelave.

V sklopu poganjanja na lokalni infrastrukturi bom aplikacijo pognal na gruči z enim, dvema in tremi vozlišči. Tako bomo lahko primerjali hitrosti glede na razpoložljive računalniške vire. Aplikacijo lahko poženemo iz kateregakoli vozlišča. BMapReduce sprejme tri argumente v ukazni vrstici. Prvi je mapa, v kateri so podatki o vremenu, drugi je mapa, v katero se rezultati obdelave zapišejo, tretji pa datoteka s podatki o postajah. Poganjanje je enako ne glede na število vozlišč v gruči. Začnimo s poganjanjem v gruči z enim vozliščem. Rezultate bom detajlno opisal samo pri prvi obdelavi, za druge bom prikazal samo ključne rezultate za primerjavo.

hsingle@hsingle:~$ hadoop jar BMapReduce.jar si.bron.BMapReduce /data/input /data/output/result270420142248 /data/stations_unique/stations.txt

14/06/10 22:49:00 INFO mapreduce.Job: Running job: job_1402432904486_0001
14/06/10 22:49:09 INFO mapreduce.Job: map 0% reduce 0%
14/06/10 22:49:31 INFO mapreduce.Job: map 1% reduce 0%
…
14/06/10 23:37:16 INFO mapreduce.Job: map 98% reduce 0%
14/06/10 23:37:34 INFO mapreduce.Job: map 99% reduce 0%
14/06/10 23:37:52 INFO mapreduce.Job: map 100% reduce 0%
14/06/10 23:38:07 INFO mapreduce.Job: map 100% reduce 100%
14/06/10 23:38:09 INFO mapreduce.Job: Job job_1402432904486_0001 completed successfully
14/06/10 23:38:09 INFO mapreduce.Job: Counters: 45
File System Counters
FILE: Number of bytes read=3350199
FILE: Number of bytes written=31389044
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=37528869030
HDFS: Number of bytes written=1115709
HDFS: Number of read operations=915
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=304
Launched reduce tasks=1
Data-local map tasks=304
Total time spent by all maps in occupied slots (ms)=2475633
Total time spent by all reduces in occupied slots (ms)=6537
Map-Reduce Framework
Map input records= 291310268
Map output records=289837332
Map output bytes=5506909308
Map output materialized bytes=3352017
Input split bytes=34352
Combine input records=289837332
Combine output records=159533
Reduce input groups=15826

Reduce shuffle bytes=3352017
Reduce input records=159533
Reduce output records=15826
Spilled Records=319066
Shuffled Maps =304
Failed Shuffles=0
Merged Map outputs=304
GC time elapsed (ms)=54076
CPU time spent (ms)=1880690
Physical memory (bytes) snapshot=81666830336
Virtual memory (bytes) snapshot=353649999872
Total committed heap usage (bytes)=63585124352
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=37528834678
File Output Format Counters
Bytes Written=1115709
si.bron.BMapper$Temperature
ARRAY_OUT_OF_BOUNDS=1
MISSING_TEMP=1472935



MapReduce Job
job_1402432904486_0001

Job Overview

Job Name:	Avg Temperature MapReduce V3
User Name:	hsingle
Queue:	default
State:	SUCCEEDED
Uberized:	false
Started:	Tue Jun 10 22:49:07 CEST 2014
Finished:	Tue Jun 10 23:38:05 CEST 2014
Elapsed:	48mins, 58sec
Diagnostics:	
Average Map Time	8sec
Average Reduce Time	1sec
Average Shuffle Time	4sec
Average Merge Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jun 10 22:49:03 CEST 2014	hsingle:8042	logs

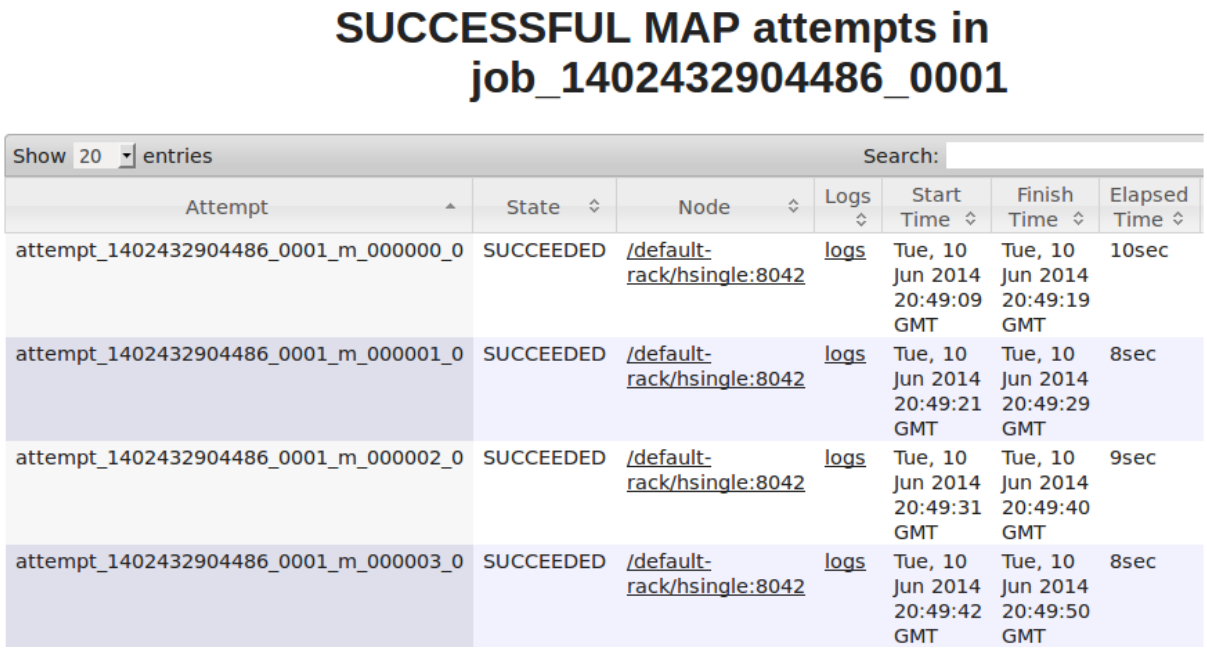
Task Type	Total	Complete
Map	304	304
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	304
Reduces	0	0	1

Slika 14: Pregled končane obdelave na enem vozlišču v brskalniku

Po končani obdelavi dobimo statistike, ki jih lahko pregledujemo preko spletnega vmesnika (glej sliki 14 in 15). Uporabimo jih lahko za optimizacijo in informacijo o tem, kaj se v gruči dogaja med obdelavo. Najzanimivejše sem odebelil. Informacije o pognanih procesih preslikovanja in krčenja, število vhodnih vrstic v vsakem koraku in nasploh število vhodnih vrstic na začetku obdelave in končno število vrstic v rezultatu nam dajejo predstavo o tem, kako velika količina podatkov se pretaka med obdelavo. Na koncu nam statistika pokaže še vrednost števcev, ki smo jih vključili v program MapReduce. Iz števca ARRAY_OUT_OF_BOUNDS vidimo, da je ena vrstica nima pričakovane vrednosti na pravem mestu. MISSING_TEMP nam pove, koliko je manjkajočih ali nepravilno oblikovanih (oz. jih ni bilo mogoče pretvoriti v številko) temperatur.

V brskalniku lahko pregledno pregledujemo tudi posamezne korake preslikovanja in krčenja, na katerih vozliščih so bili pognani in koliko časa so trajali (glej sliko 15). Enostavno lahko vidimo, katero vozlišče je bilo izbrano kot skrbnik aplikacije (glej sliko 14, drugi razdelek). Brskalnik kot pregledovanje obdelav je zelo priročen, saj se nam ni potrebno prebijati skozi več sto log datotek.



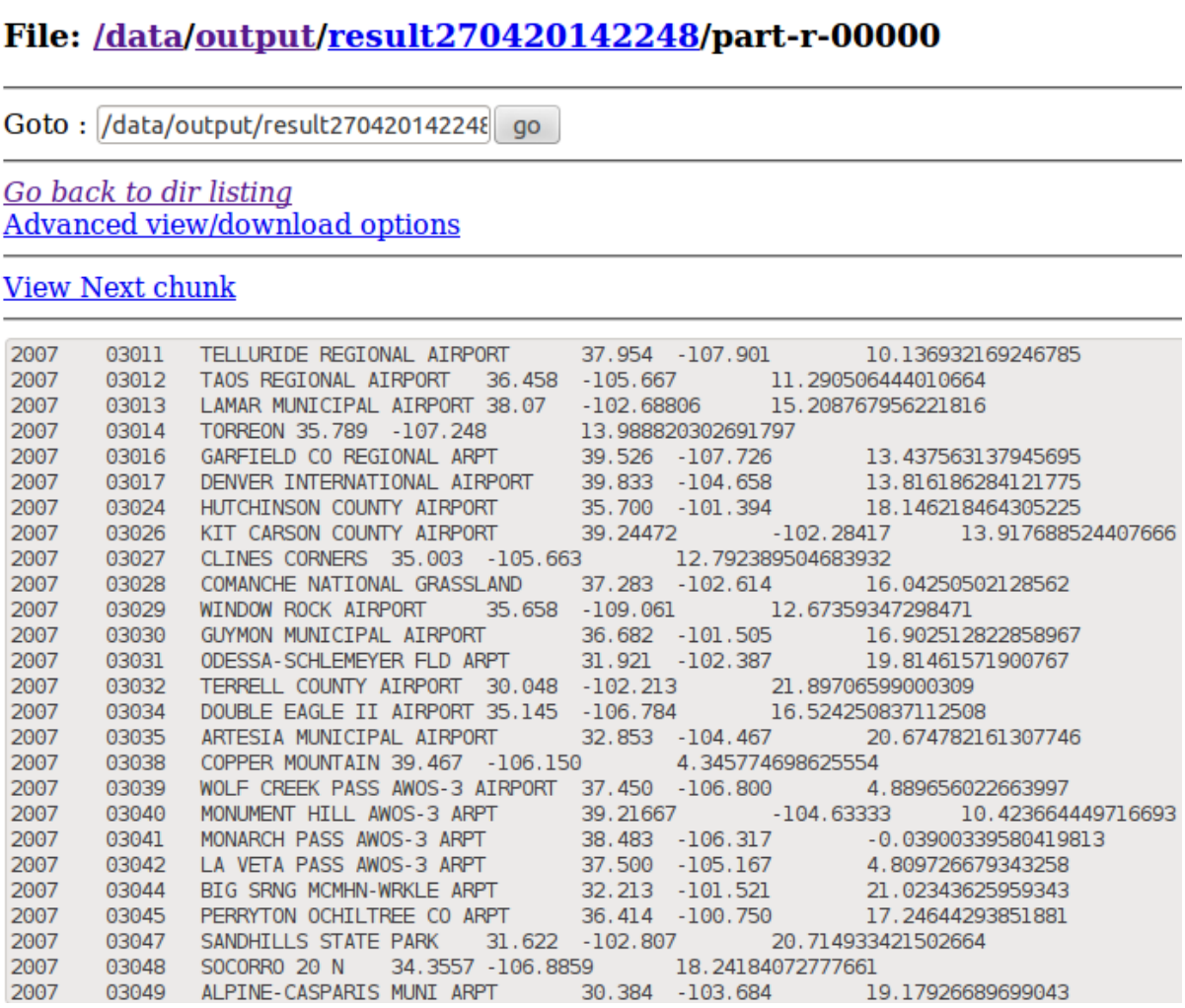
SUCCESSFUL MAP attempts in job_1402432904486_0001

Show 20 entries Search:

Attempt	State	Node	Logs	Start Time	Finish Time	Elapsed Time
attempt_1402432904486_0001_m_000000_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Tue, 10 Jun 2014 20:49:09 GMT	Tue, 10 Jun 2014 20:49:19 GMT	10sec
attempt_1402432904486_0001_m_000001_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Tue, 10 Jun 2014 20:49:21 GMT	Tue, 10 Jun 2014 20:49:29 GMT	8sec
attempt_1402432904486_0001_m_000002_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Tue, 10 Jun 2014 20:49:31 GMT	Tue, 10 Jun 2014 20:49:40 GMT	9sec
attempt_1402432904486_0001_m_000003_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Tue, 10 Jun 2014 20:49:42 GMT	Tue, 10 Jun 2014 20:49:50 GMT	8sec

Slika 15: Izsek pregleda končanih korakov preslikovanja na vozlišču

Obdelava na enem vozlišču je trajala 48 minut in 58 sekund. Rezultat vsebuje 15826 vrstic, kar je enako kot število izhodnih vrstic krčenja (glej začetek 4.1, kjer so tekstovni rezultati obdelave). Rezultat obdelave se je shranil v /data/output/result270420142248. HDFS v tej mapi ustvari toliko izhodnih datotek, kolikor je korakov krčenja. V mojem primeru je to ena datoteka s privzetim imenom »part-r-00000« (glej sliko 16).



File: /data/output/result270420142248/part-r-00000

Goto : /data/output/result270420142248 go

Go back to dir listing
Advanced view/download options

View Next chunk

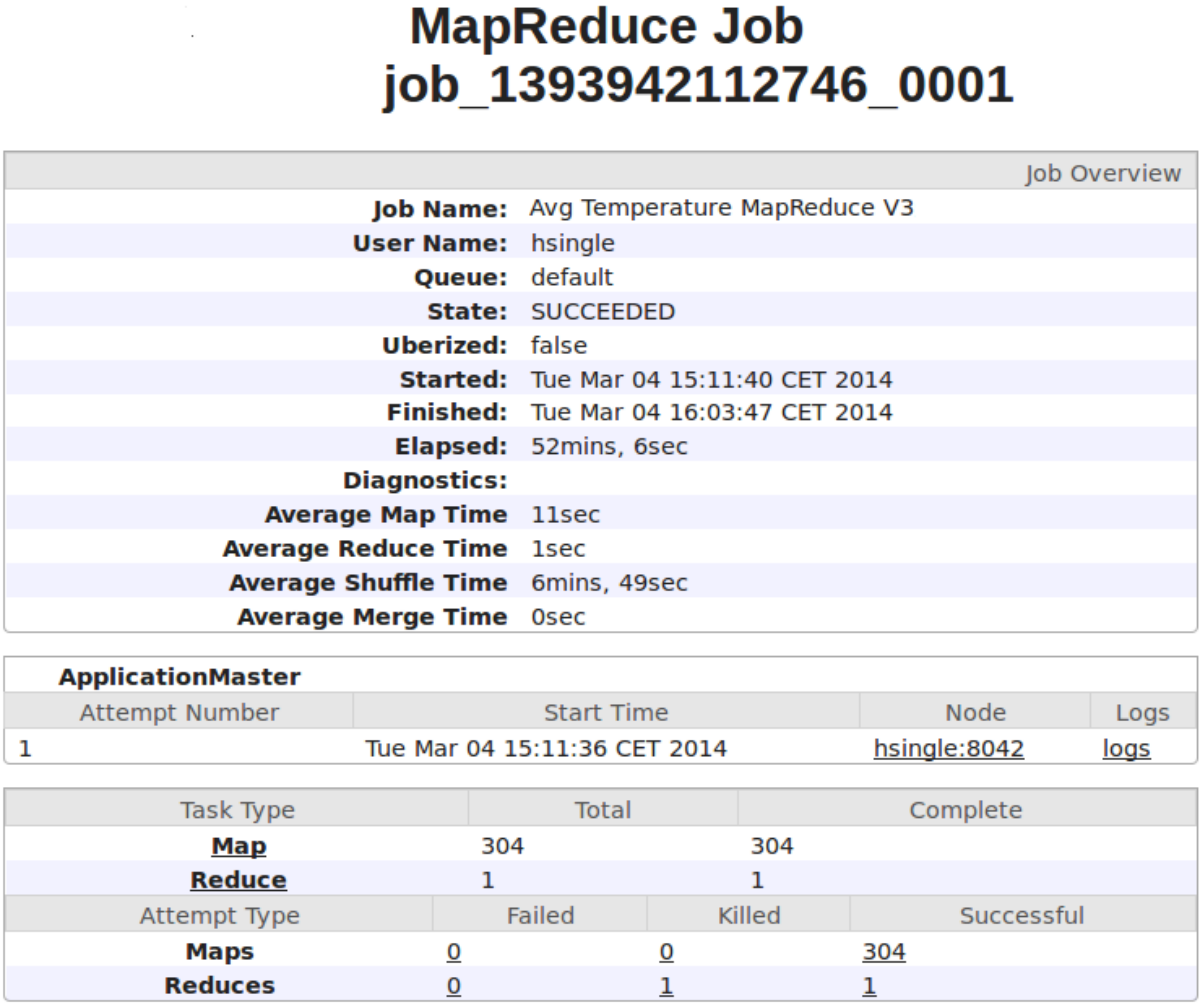
```
2007    03011   TELLURIDE REGIONAL AIRPORT      37.954  -107.901        10.136932169246785
2007    03012   TAOS REGIONAL AIRPORT   36.458  -105.667        11.290506444010664
2007    03013   LAMAR MUNICIPAL AIRPORT 38.07   -102.68806      15.208767956221816
2007    03014   TORREON 35.789  -107.248        13.988820302691797
2007    03016   GARFIELD CO REGIONAL ARPT       39.526  -107.726        13.437563137945695
2007    03017   DENVER INTERNATIONAL AIRPORT    39.833  -104.658        13.816186284121775
2007    03024   HUTCHINSON COUNTY AIRPORT       35.700  -101.394        18.146218464305225
2007    03026   KIT CARSON COUNTY AIRPORT       39.24472        -102.28417      13.917688524407666
2007    03027   CLINES CORNERS  35.003  -105.663        12.792389504683932
2007    03028   COMANCHE NATIONAL GRASSLAND     37.283  -102.614        16.04250502128562
2007    03029   WINDOW ROCK AIRPORT     35.658  -109.061        12.67359347298471
2007    03030   GUYMON MUNICIPAL AIRPORT        36.682  -101.505        16.902512822858967
2007    03031   ODESSA-SCHLEMEYER FLD ARPT      31.921  -102.387        19.81461571900767
2007    03032   TERRELL COUNTY AIRPORT  30.048  -102.213        21.89706599000309
2007    03034   DOUBLE EAGLE II AIRPORT 35.145  -106.784        16.524250837112508
2007    03035   ARTESIA MUNICIPAL AIRPORT       32.853  -104.467        20.674782161307746
2007    03038   COPPER MOUNTAIN 39.467  -106.150        4.345774698625554
2007    03039   WOLF CREEK PASS AWOS-3 AIRPORT  37.450  -106.800        4.889656022663997
2007    03040   MONUMENT HILL AWOS-3 ARPT       39.21667        -104.63333      10.423664449716693
2007    03041   MONARCH PASS AWOS-3 ARPT        38.483  -106.317        -0.03900339580419813
2007    03042   LA VETA PASS AWOS-3 ARPT        37.500  -105.167        4.809726679343258
2007    03044   BIG SRNG MCMHN-WRKLE ARPT       32.213  -101.521        21.02343625959343
2007    03045   PERRYTON OCHILTREE CO ARPT      36.414  -100.750        17.24644293851881
2007    03047   SANDHILLS STATE PARK    31.622  -102.807        20.714933421502664
2007    03048   SOCORRO 20 N    34.3557 -106.8859       18.24184072777661
2007    03049   ALPINE-CASPARIS MUNI ARPT       30.384  -103.684        19.17926689699043
```

Slika 16: Skrajšan rezultat obdelave prikazan v brskalniku

Rezultat obdelave je vsebinsko pravilen. Datoteka vsebuje leto, šifro WBAN, ime postaje, zemljepisno širino, zemljepisno dolžino ter povprečno temperaturo za leto. Vrstice so sortirane naraščajoče po letu in šifri WBAN. Za privzeto sortiranje po ključu, v tem primeru sta ključ leto in šifra WBAN poskrbi MapReduce korak premetavanja.

Datoteko lahko poljubno shranimo preko brskalnika ali preko ukazov HDFS (npr. hdfs dfs –copyToLocal <ime datoteke> <pot na lokalnem podatkovnem sistemu>). Datoteko bi lahko uporabili kot vir za Hive, Hbase, RDBMS (*angl. Relational Database Management System*) ipd.

Obdelavo MapReduce poženem na gruči dveh vozlišč. V tej gruči sta vozlišči hsingle in hsingle2. Hsingle2 je močno podhranjeno vozlišče kar se tiče računalniških virov, saj ima na razpolago samo eno procesorsko jedro in 1536 MB pomnilnika.



MapReduce Job
job_1393942112746_0001

Job Overview	
Job Name:	Avg Temperature MapReduce V3
User Name:	hsingle
Queue:	default
State:	SUCCEEDED
Uberized:	false
Started:	Tue Mar 04 15:11:40 CET 2014
Finished:	Tue Mar 04 16:03:47 CET 2014
Elapsed:	52mins, 6sec
Diagnostics:	
Average Map Time	11sec
Average Reduce Time	1sec
Average Shuffle Time	6mins, 49sec
Average Merge Time	0sec

ApplicationMaster

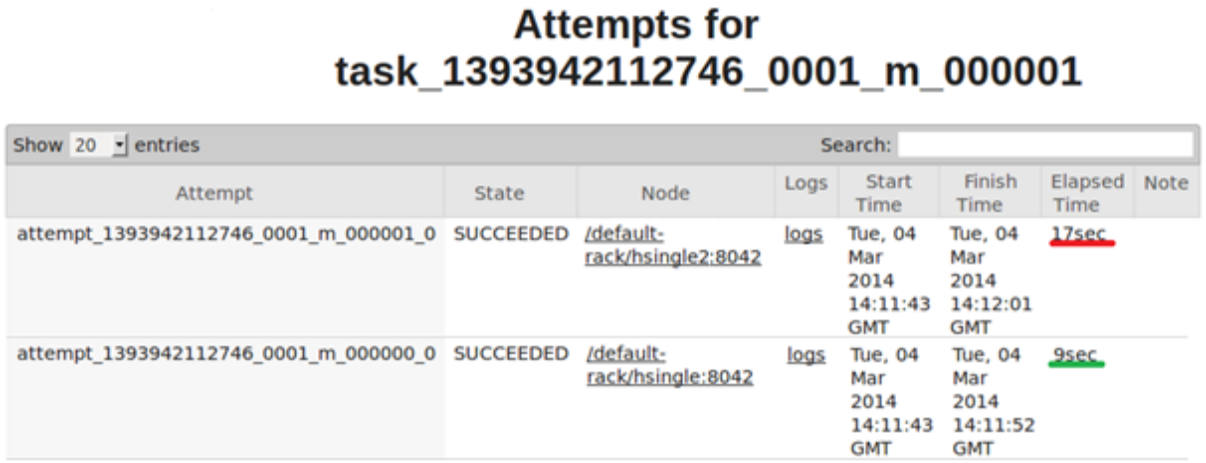
Attempt Number	Start Time	Node	Logs
1	Tue Mar 04 15:11:36 CET 2014	hsingle:8042	logs

Task Type	Total	Complete
Map	304	304
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	304
Reduces	0	1	1

Slika 17: Pregled končane obdelave na gruči dveh vozliščih v brskalniku

Obdelava na dveh vozliščih je trajala 52 minut in 6 sekund (glej sliko 17), kar je 3 minute in 8 sekund dlje. Razlog je v tem, da je hsingle2 močno podhranjen. S 1536MB spomina namenjenega obdelavi in prenosom preko omrežja izgubljamo čas namesto da bi ga pridobili. Poleg tega je hitrost vrtenja diska na hsingle2 le 5400 obratov na minuto, medtem ko je na hsingle 7200. To nekoliko škoduje operacijam branja in pisanja. Na spodnji sliki (glej sliko 18) se vidi, da so koraki preslikovanja na hsingle2 za polovico počasnejši. Na sliki 18 je prikazano, da je povprečen korak preslikovanja na hsingle2 trajal kar 17 sekund na hsingle pa 9 sekund. Če bi obdelavo pognali samo na vozlišču hsingle2, bi obdelava trajala skoraj dvakrat dlje kot na vozlišču hsingle. Edini razlog, da ni vozlišče hsingle2 še več vplivalo na trajanje obdelave, je po mojem mnenju v tem, da je hsingle2 obdelalo manj korakov preslikovanja kot vozlišče hsingle, saj je skrbnik aplikacije vozlišču hsingle hitreje dodeljevalo preslikave in to je vplivalo na število preslikav obdelanih na posameznih vozliščih.



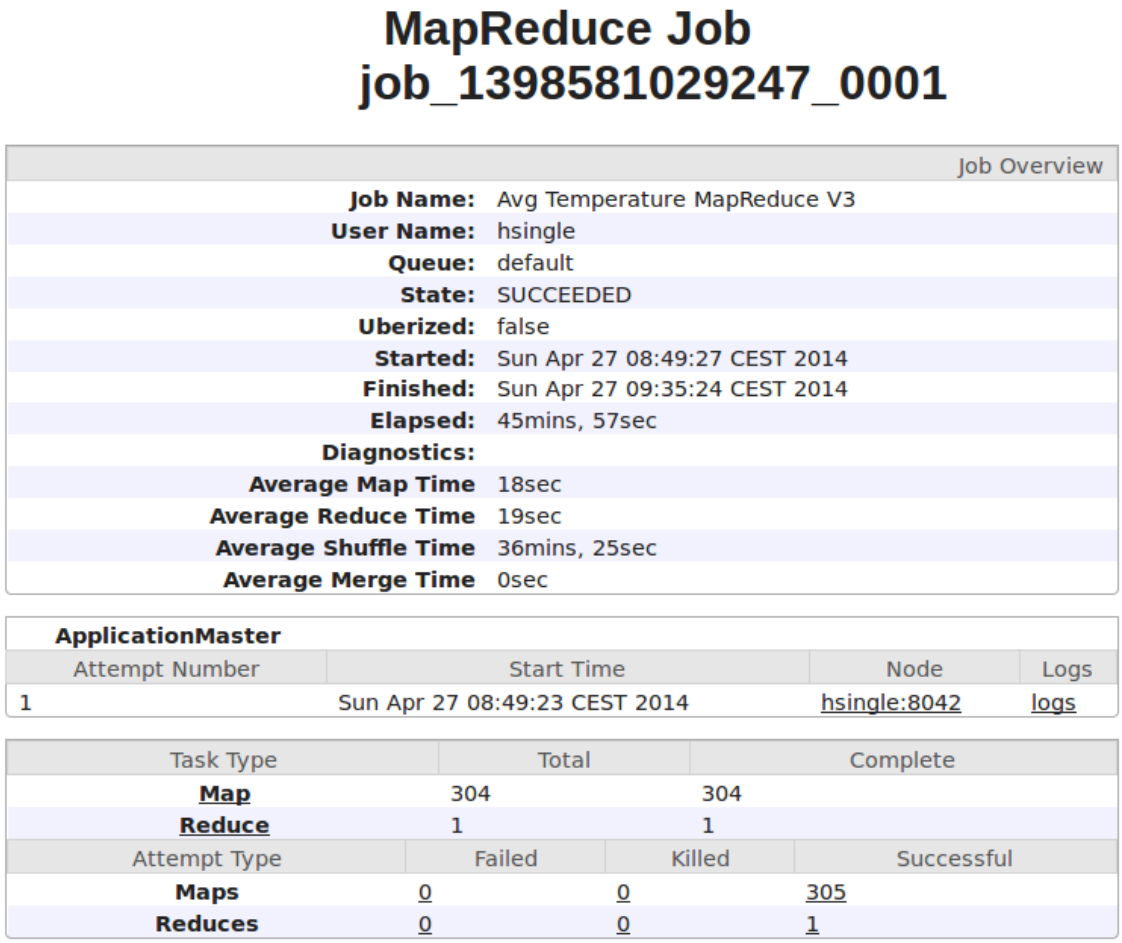
Attempts for task_1393942112746_0001_m_000001

Show 20 entries Search:

Attempt	State	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1393942112746_0001_m_000001_0	SUCCEEDED	/default-rack/hsingle2:8042	logs	Tue, 04 Mar 2014 14:11:43 GMT	Tue, 04 Mar 2014 14:12:01 GMT	17sec	
attempt_1393942112746_0001_m_000000_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Tue, 04 Mar 2014 14:11:43 GMT	Tue, 04 Mar 2014 14:11:52 GMT	9sec	

Slika 18: Izsek iz pregleda korakov preslikovanja gruče dveh vozlišč

Za dodatno primerjavo poženem obdelavo še na gruči treh vozlišč (glej sliki 19 in 20).



MapReduce Job job_1398581029247_0001

Job Overview	
Job Name:	Avg Temperature MapReduce V3
User Name:	hsingle
Queue:	default
State:	SUCCEEDED
Uberized:	false
Started:	Sun Apr 27 08:49:27 CEST 2014
Finished:	Sun Apr 27 09:35:24 CEST 2014
Elapsed:	45mins, 57sec
Diagnostics:	
Average Map Time	18sec
Average Reduce Time	19sec
Average Shuffle Time	36mins, 25sec
Average Merge Time	0sec

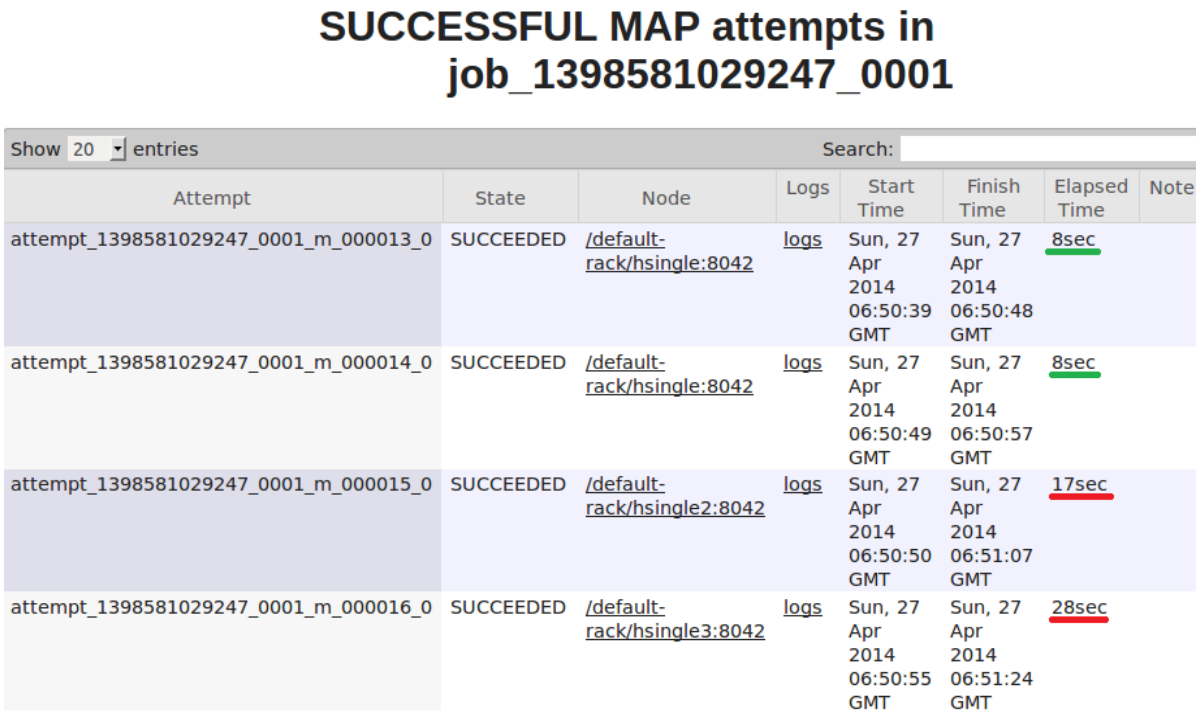
ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Apr 27 08:49:23 CEST 2014	hsingle:8042	logs

Task Type	Total	Complete
Map	304	304
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	305
Reduces	0	0	1

Slika 19: Pregled končane obdelave na gruči treh vozliščih v brskalniku



SUCCESSFUL MAP attempts in job_1398581029247_0001

Show 20 entries Search:

Attempt	State	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1398581029247_0001_m_000013_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Sun, 27 Apr 2014 06:50:39 GMT	Sun, 27 Apr 2014 06:50:48 GMT	8sec	
attempt_1398581029247_0001_m_000014_0	SUCCEEDED	/default-rack/hsingle:8042	logs	Sun, 27 Apr 2014 06:50:49 GMT	Sun, 27 Apr 2014 06:50:57 GMT	8sec	
attempt_1398581029247_0001_m_000015_0	SUCCEEDED	/default-rack/hsingle2:8042	logs	Sun, 27 Apr 2014 06:50:50 GMT	Sun, 27 Apr 2014 06:51:07 GMT	17sec	
attempt_1398581029247_0001_m_000016_0	SUCCEEDED	/default-rack/hsingle3:8042	logs	Sun, 27 Apr 2014 06:50:55 GMT	Sun, 27 Apr 2014 06:51:24 GMT	28sec	

Slika 20: Izsek iz pregleda korakov preslikovanja gruče treh vozlišč

Obdelava na treh vozliščih je bila le za 3 minute in 1 sekundo hitrejša od obdelave na enem vozlišču. Znova je vozlišče hsingle nalogo opravljalo najbolje. V času obdelave enega preslikovanja obeh preostalih vozlišč je hsingle obdelal dva. Opazimo, da je dodatno vozlišče hsingle3 najslabše opravljalo delo. Hsingle3 ima 1024 MB spomina rezerviranega za obdelave. Poleg tega pa je za dodatno povečanje časa obdelave kriva tudi brezžična povezava. Kljub temu da je v zadnji obdelavi bil povprečen čas preslikovanj daljši, je gruča lahko obdelovala 2-3 preslikovanja istočasno. V prejšnjih primerih pa 1-2.

Kljub enostavni namestitvi gruče Hadoop je optimizacija le-te zahtevna in odvisna od strojne opreme vozlišč. Vse skupaj bi se bolje obnašalo, če bi imel na voljo tri vozlišča z najmanjšo specifikacijo 4 GB spomina, 4 procesorskia jedra in disk s hitrostjo 7200 obratov na minuto, morda celo SSD (*angl. Solid-State Drive*). Poleg naštetega bi pripomogel tudi boljši usmerjevalni, ki bi poleg boljše prepustnosti (npr. 1Gbit na sekundo ali več) podpiral povezavo vseh vozlišč preko kabla.

Poleg strojne opreme uporabljene v gruči je zelo pomembno mikro nastavljanje posameznih komponent (YARN, HDFS), poleg nujnih nastavitev, ki so opisane v poglavju 3.5, obstaja še mnogo drugih specifičnih nastavitev. V teh nastavitvah med drugim lahko določimo, koliko spomina bo posamezen proces, zagnan na vozlišču, porabil in koliko procesov lahko nastopa

istočasno. Z 1 GB spomina tega manevrskega prostora ni. Povzetek časov obdelav je prikazan v tabeli 5.

Gruča	Čas obdelave
1 vozlišče	48 min 58 sek
2 vozlišči	52 min 6 sek
3 vozlišča	45 min 57 sek

Tabela 5: Povzetek časov obdelav na lokalnih gručah

4.2 Poganjanje BMapReduce v oblaku

Za poganjanje v oblaku je izbran Amazon EMR. Obdelavo bom poganjal na osmih različnih gručah. Obdelavo na gruči 10 velikih vozlišč bom natančneje opisal in prikazal uporabo vmesnika EMR. Pri drugih bom samo prikazal rezultate za primerjavo.

Specifikacija strojne opreme vozlišč v gručah. Vozlišča razdelim na srednja, velika in zelo velika:

- srednje: 2 GB spomina za obdelave, eno virtualno procesorsko jedro, disk 410 GB
- veliko: 5 GB spomina za obdelave, dve virtualni procesorski jedri, disk 840 GB
- zelo veliko: 32 GB spomina za obdelave, osem virtualnih procesorskih jeder, disk 1,68 TB

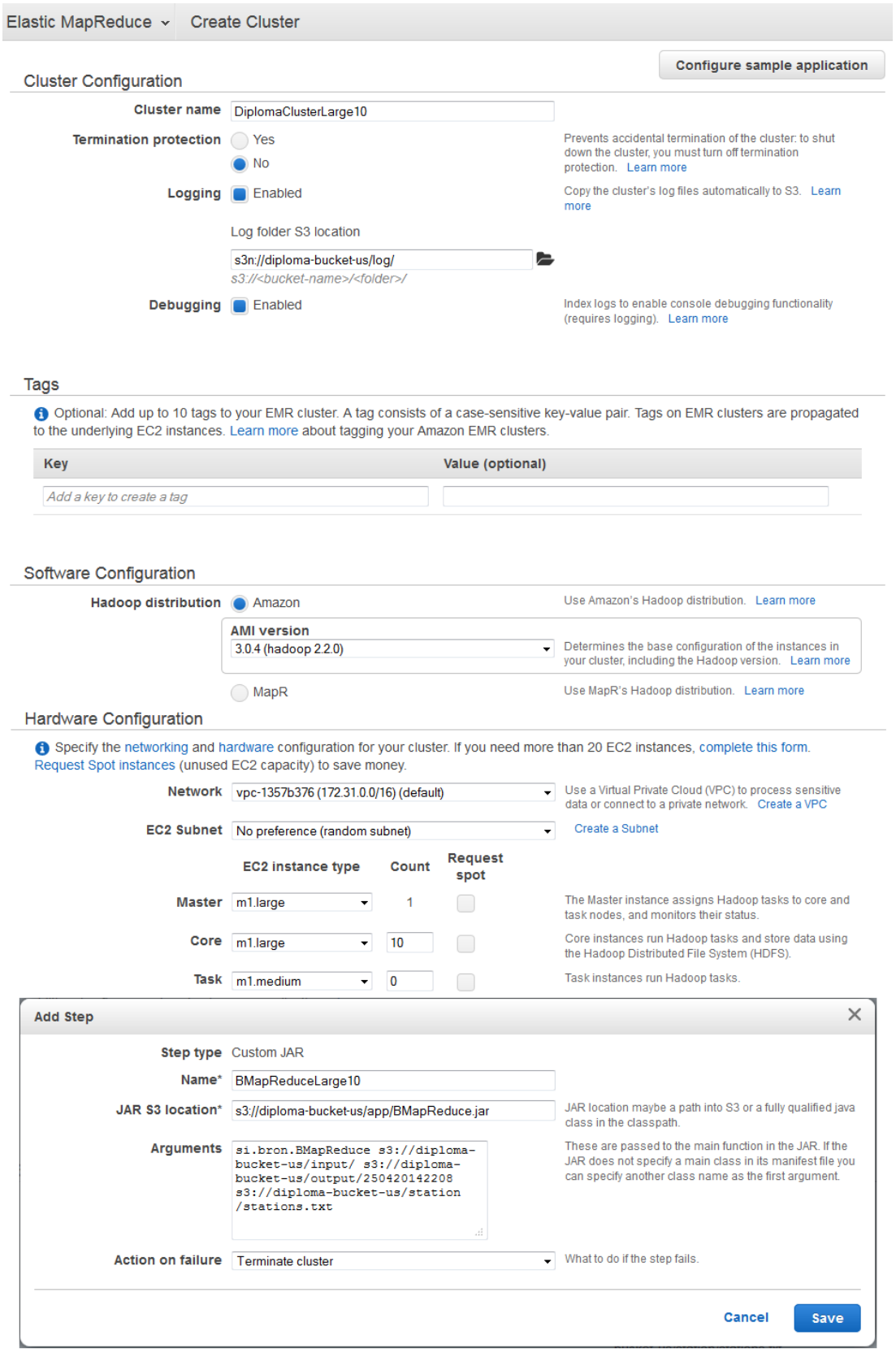
Specifikacija gruč:

- 3 srednja vozlišča s podatki v S3
- 5 srednjih vozlišč s podatki v S3
- 10 srednjih vozlišč s podatki v S3
- 5 velikih vozlišč s podatki v S3
- 10 velikih vozlišč s podatki v S3
- 10 zelo velikih vozlišč s podatki v S3
- 3 srednja vozlišča s podatki v HDFS
- 10 zelo velikih vozlišč s podatki v HDFS

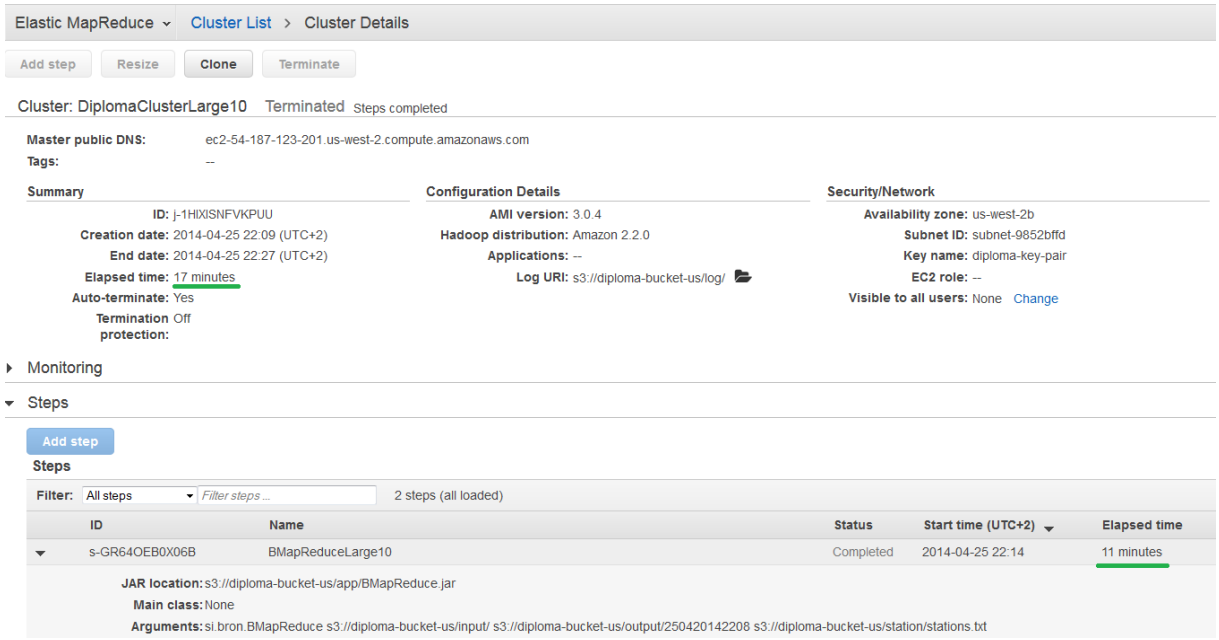
Obdelave v oblaku Amazon EMR lahko uporabljajo različne podprte datotečne sisteme (npr. S3, HDFS, datotečni sistem operacijskega sistema virtualnega vozlišča). Odločil sem se večinoma za S3, saj je enostavnejši za uporabo, vendar počasnejši od HDFS, saj podatki niso razpršeni po vseh podatkovnih vozliščih gruče. Pravtako v S3 ni mogoče zagotavljati lokalnosti obdelav glede na podatke kar občutno poveča čas obdelave [4]. Za potrebe primerjave bom na koncu predstavil tudi poganjanje v HDFS načinu.

Zagnati novo EMR gručo in pognati obdelavo je enostavno. Potrebno je določiti ime gruče, opredeliti število in tip vozlišč ter verzijo Hadoop. Na koncu pa dodati prvi korak po vzpostavitvi gruče, ki bo pognal obdelavo BMapReduce. Vhodni parametri v obdelavi so sedaj enake datoteke in mape kot pri lokalnem poganjanju naložene v S3. Slika 21 prikazuje nastavljanje parametrov gruče. Ko potrdimo nastavitve gruče, se začne namestitev gruče na Amazon EMR in požene prva obdelava BMapReduceLarge10. Gručo sem nastavil tako, da se po obdelavi ugasne.

Na sliki lahko vidimo, da za poganjanje obdelav v gruči 10 vozlišč moramo dejansko najetei 11 vozlišč. Amazon namreč loči glavno vozlišče od podrejenih vozliščih. Obdelave potekajo samo na računalniških virih podrejenih vozliščih. Amazon glavno vozlišče prevzema delo glavnega vozlišča HDFS in upravitelja virov za YARN. Za enostavnost, bom pri vseh izračunih in primerjavah kasneje upošteval samo podrejena vozlišča v oblaku.



Slika 21: Primer nastavitev gruče EMR



Slika 22: Končana obdelava na gruči 10 velikih vozlišč

Na sliki 22 sta označena dva podatka, prvi je celoten čas, ki ga je porabil Amazon EMR za postavitev gruče in poganjanje obdelave, drugi pa trajanje obdelave same. Obdelava je bila zaključena v enajstih minutah. Sledi tabela 6, ki prikazuje čas obdelave za gruče, ki imajo podatke naložene v S3.

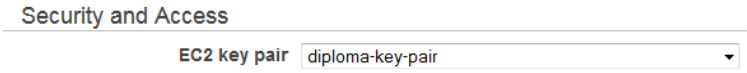
Gruča	Čas obdelave (min)
3 srednja vozlišča (najbolj podobna lokalni gruči)	100 min
5 srednjih vozlišč	49 min
10 srednjih vozlišč	24 min
5 velikih vozlišč	21 min
10 velikih vozlišč	11 min
10 zelo velikih vozlišč	3 min

Tabela 6: Pregled časov izvajanja v gručah v oblaku

Za zaključek poglavja bom prikazali razliko med poganjanjem obdelave v oblaku s podatki v S3 in HDFS, ki ga imajo vozlišča nameščenega lokalno. Način prenosa podatkov v HDFS je bolj zapleten kot prenos podatkov v S3 in uporabniko brez tehničnega znanja lahko povzroča kar nekaj preglavic.

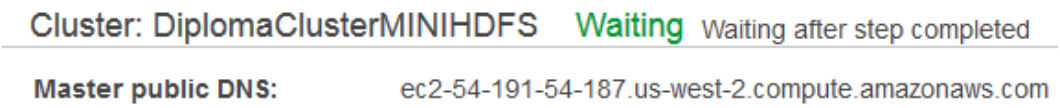
Amazon ponuja povezave na svoje strežnike preko SFTP (*angl. Secure File Transfer Protocol*) in SSH (*angl. Secure Shell*) protokolov. Za povezavo je potrebno generirati ključ, ki

ga lahko pridobimo preko Amazonove uporabniške konzole na spletu. Poženemo gručo treh srednjih vozlišč vendar moramo paziti, da izberemo nastavitev povezljivosti preko ključa (glej sliko 23).



Slika 23: Prikaz komponent izbire ključa pri nastavljanju gruče

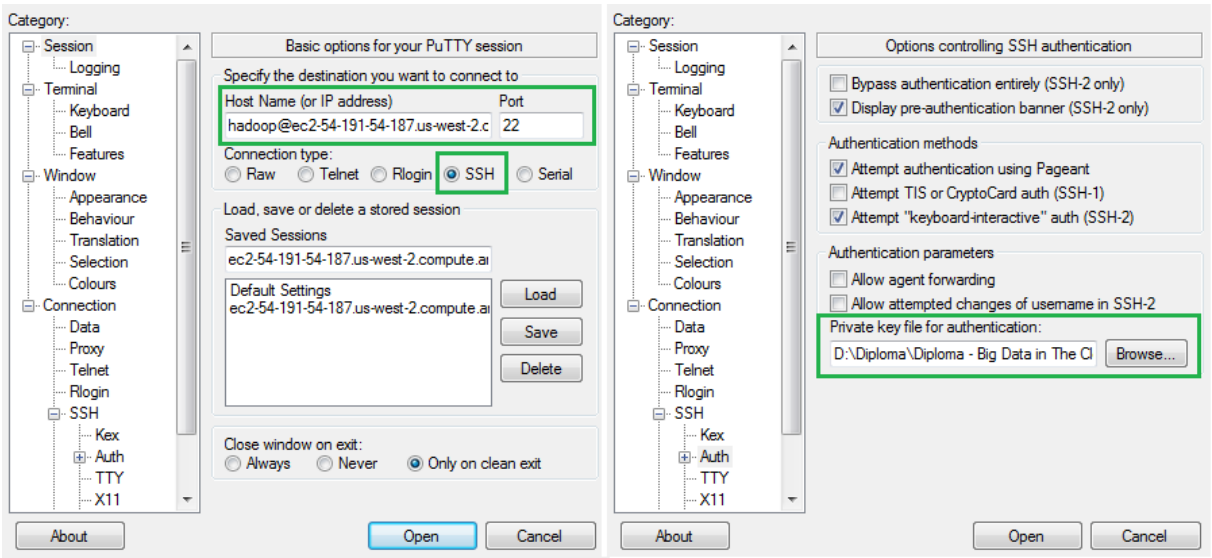
Ko se gruča postavi, dobimo informacijo o DNS (*angl. Domain Name System*) naslovu glavnega vozlišča (glej sliko 24). Potrebovali ga bomo pri povezovanju preko SSH in SFTP.



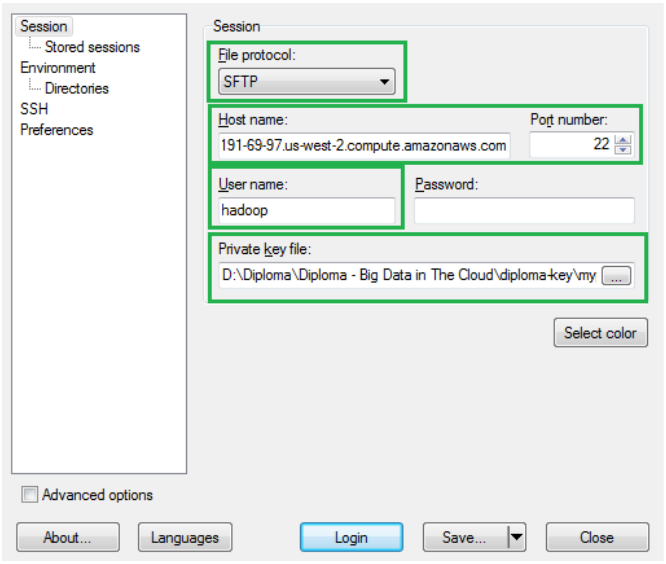
Slika 24: Gruča v stanju pripravljenosti in DNS glavnega vozlišča

Ključ iz Amazon uporabniške konzole prenesemo na računalnik iz katerega bomo dostopali do gruče, saj ga je potrebno uporabiti pri povezavi. Sam sem uporabil Putty in WinSCP kot orodji za dostopanje do gruče.

V Putty in WinSCP povezavo nastavimo takole (glej sliki 25 in 26):



Slika 25: Putty povezava do glavnega vozlišča



Slika 26: WinSCP povezava do glavnega vozlišča

Sedaj imamo pripravljene povezave na glavno vozlišče, kjer bomo poganjali obdelave. Preko Putty poženemo naslednje ukaze in pripravimo okolje:

- hdfs dfs -mkdir /data
- hdfs dfs -mkdir /data/input
- hdfs dfs -mkdir /data/output
- hdfs dfs -mkdir /data/stations
- hadoop distcp s3n://diploma-bucket-us/input/*.* /data/input
- hadoop distcp s3n://diploma-bucket-us/station/stations.txt /data/stations/stations.txt

Prvi štirje ukazi so namenjeni kreiranju map v HDFS. Zadnja dva pa za prenos podatkov o vremenu in informacije o postajah iz S3 v HDFS. Prenos 35 GB podatkov se zaključi v nekaj minutah, saj je povezava med infrastrukturo v oblaku zelo hitra. Preko WinSCP kopiramo BMapReduce.jar na glavno vozlišče v mapo /home/hadoop. Okolje je pripravljeno. Prav tako kot v lokalnem okolju, lahko do HDFS dostopamo preko brskalnika na privzetih vratih 9101 (glej sliko 27).

Started:	Tue Jul 01 11:31:22 UTC 2014
Version:	2.2.0, 464fbe2a1346c21817571cda8a65c0c4cf54a081
Compiled:	2014-02-17T22:01Z by Elastic MapReduce from (detached from 464fbe2)
Cluster ID:	CID-c56af530-0871-4ffe-a52f-5eb23e634009
Block Pool ID:	BP-119242595-172.31.38.170-1404214248899

Browse the filesystem
NameNode Logs

Cluster Summary

Security is *OFF*
101 files and directories, 316 blocks = 417 total.
Heap Memory used 30.84 MB is 54% of Commited Heap Memory 56.81 MB. Max Heap Memory is 966.69 MB.
Non Heap Memory used 28.11 MB is 95% of Commited Non Heap Memory 29.44 MB. Max Non Heap Memory is 130 MB.

Configured Capacity	:	1.17 TB			
DFS Used	:	35.33 GB			
Non DFS Used	:	0 B			
DFS Remaining	:	1.14 TB			
DFS Used%	:	2.95%			
DFS Remaining%	:	97.05%			
Block Pool Used	:	35.33 GB			
Block Pool Used%	:	2.95%			
DataNodes usages	:	Min %	Median %	Max %	stdev %
		2.76%	3.04%	3.05%	0.13%
Live Nodes	:	3 (Decommissioned: 0)			
Dead Nodes	:	0 (Decommissioned: 0)			
Decommissioning Nodes	:	0			
Number of Under-Replicated Blocks	:	0			

Slika 27: Prikaz statistike HDFS gruče preko brskalnika. Na sliki med drugim vidimo privzeto kapaciteto diskov gruče treh srednjih vozlišč

Ostane nam samo še poganjanje obdelave. To storimo preko Putty konzole z naslednjim ukazom:

[hadoop@ip-172-31-38-170 ~]$ hadoop jar BMapReduce.jar si.bron.BMapReduce /data/input /data/output/resultRemoteHDFS /data/stations/stations.txt

Obdelava, ki ima podatke v HDFS je po pričakovanjih hitrejša od obdelave s podatki v S3 (glej sliko 28).

Razlika v času obdelave je ogromna. V HDFS načinu je bila obdelava za kar 53 minut hitrejša kot S3. Kot zanimivost sem enake korake ponovil na gruči 10 zelo velikih vozlišč, ker me je zanimalo koliko časa lahko pridobim. Razlika ni toliko očitna kot na manj zmogljivi gruči, je pa proporcionalno primerljiva, saj enako kot obdelava na manjši gruči se tudi na veliki gruči

čas zmanjša za polovico. Statistiko obdelave prikazuje slika 29. Tabela 7 prikazuje čase izvajanja obdelav pognanih na gručah, ki imajo podatke v HDFS.

Application Overview

User:	hadoop
Name:	Avg Temperature MapReduce V3
Application Type:	MAPREDUCE
State:	FINISHED
FinalStatus:	SUCCEEDED
Started:	1-Jul-2014 12:32:11
Elapsed:	47mins, 25sec
Tracking URL:	History
Diagnostics:	

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	1-Jul-2014 12:32:11	ip-172-31-40-140.us-west-2.compute.internal:9035	logs

Slika 28: Pregled končane obdelave na gruči 3 srednjih vozlišč v oblaku s podatki v HDFS

Application Overview

User:	hadoop
Name:	Avg Temperature MapReduce V3
Application Type:	MAPREDUCE
State:	RUNNING
FinalStatus:	UNDEFINED
Started:	1-Jul-2014 12:32:11
Elapsed:	1mins, 36sec
Tracking URL:	ApplicationMaster
Diagnostics:	

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	1-Jul-2014 12:32:11	ip-172-31-40-140.us-west-2.compute.internal:9035	logs

Slika 29: Pregled končane obdelave na gruči 10 zelo velikih vozlišč v oblaku s podatki v HDFS

Poglavje 5 Primerjave

Zadnji cilj diplomske naloge je primerjava poganjanja v oblaku in lokalni gruči. Primerjal bom trajanje obdelav, ceno in čas potreben za namestitev lokalnega okolja in poganjanje obdelave v oblaku. Kjer se kot strošek pojavlja čas dela človeka uporabim urno postavko 35 EUR. Pripravil sem naslednje primerjave:

- Časovna in cenovna primerjava gruč v oblaku
- Časovna primerjava med izvajanjem v oblaku in lokalni infrastrukturi
- Cenovna primerjava med izvajanjem ene obdelave v oblaku in mojem lokalnem okolju
- Cenovna primerjava med najemom gruče treh srednjih vozlišč za konstantno izvajanje obdelav v oblaku in nakupom primerljive infrastrukture za lokalno uporabo
- Cenovna primerjava med najemom gruče 10 srednjih vozlišč za konstantno izvajanje obdeav v oblaku in nakupom primerljive infrastrukture za lokalno uporabo

Preden začnemo s primerjavami si poglejmo skupne stroške obdelav (glej tabelo 7) in cenik izbranih storitev v oblaku (glej tabelo 8).

Aktivnost	Čas (h)	Cena (EUR)
Skupne aktivnosti		
Analiza zahtev	~ 2	~ 70
Učenje MapReduce programskega modela	~ 4	~ 140
Učenje Pig Latin osnov	~ 2	~ 70
Implementacija Java MapReduce aplikacije	~ 8	~ 280
Implementacija Pig Latin skripte	~ 2	~ 70
Testiranje Pig Latin skripte	~ 2	~ 70
Testiranje Java MapReduce aplikacije	~ 3	~ 105
SKUPAJ:	**~ 23**	**~ 805**

Tabela 7: Prikaz skupnih aktivnosti po času in stroških

Pri obeh načinih poganjanja se pojavlja strošek implementacije MapReduce obdelave in skripte Pig Latin. Delimo ga lahko na tri dele. Prvi je strošek učenja MapReduce modela, pripadajočih Java knjižnic in Pig Latin skriptnega jezika. Drugi strošek je povezan z vsebinsko zahtevo obdelave, pisanjem Java programske kode in Pig Latin skripte. Tretji pa s pisanjem testnih primerov in testiranjem.

Poleg tega, je obdelavam v oblaku še skupni mesečni strošek hranjenja podatkov v S3, ki za 35 GB znaša 0,77 EUR.

Storitev	Enota	Cena (EUR)
Najem infrastrukture		
Srednje vozlišče	1h/vozlišče	0,109
Veliko vozlišče	1h/vozlišče	0,21
Zelo veliko vozlišče	1h/vozlišče	0,897
Podatki		
Pretok podatkov v oblak	1GB	Vedno 0
Pretok podatkov iz oblaka	1GB	0,0001
Pretok podatkov po zahtevah (PUT,COPY,POST,LIST)	1000 zahtev	0,004
Hranjenje podatkov v oblaku	1GB	0,022

Tabela 8: Prikaz cenika izbranih storitev v oblaku [2,3]

5.1 Časovna in cenovna primerjava izbranih gruč v oblaku

Gruča	Čas obdelave	Cena poganjanja ene obdelave
3 srednja vozlišča (S3)	100 min	0,654
5 srednjih vozlišč (S3)	49 min	0,545
3 srednja vozlišča (HDFS)	47 min 25 sek	0,654
10 srednjih vozlišč (S3)	24 min	1,09
5 velikih vozlišč (S3)	21 min	1,05
10 velikih vozlišč (S3)	11 min	2,1
10 zelo velikih vozlišč (S3)	3 min	8,97
10 zelo velikih vozlišč (HDFS)	1min 36 sek	8,97

Tabela 9: Pregled hitrosti izvajanja v oblaku in cene obdelave glede na gručo

Tabela 9 prikazuje stroške poganjanja ene obdelave. V teh stroških niso všteti skupni stroški iz tabele 6 ter mesečni stroški hranjenja podatkov v S3.

Na prvi pogled tabele 9 vidimo, da so gruče, ki uporabljajo S3 počasnejše od gruč, ki uporabljajo HDFS. Razlog za to najdemo v funkcionalnosti HDFS, ki obdelave približa podatkom [4]. Če hitrost obdelave ni pomembna je S3 boljša izbira za namenske obdelave saj se ni potrebno ukvarjati z dostopom do infrastrukture v oblaku in poznavanjem ukazov HDFS. Podobno kot pri izbiri shrambe podatkov, če hitrost obdelave ni pomembna je boljša izbira manjša gruča vendar je dobro obdelave zaključiti znotraj ene ure, saj se nam ob začetku nove obdelava dvojno zaračuna. Ta pojav opazimo pri primerjavi prvih dveh gruč v tabeli. Čas obdelave je odvisen od števila vozlišč oz. razpoložljivih virov gruče in načina shrambe podatkov.

V tabeli 8 opazimo, da je cena enega poganjanja obdelave za tri sredna vozlišča z uporabo S3 in HDFS enaka. Razlog je v tem, da porabimo več časa za pravilno vzpostavitev SSH in SCP povezave in nalaganje podatkov v HDFS. Gruča med tem časom deluje in zaračunava ceno začete ure. Za vzpostavitev povezave v oblak in prenos podatkov v HDFS sem porabil 20 minut to je pa pripomoglo k višji ceni, saj je Amazon zaračunal ceno dveh začetih ur.

5.2 Časovna primerjava med izvajanjem v oblaku in lokalni infrastrukturi

V oblaku sem pognal dve gruči treh srednjih vozlišč. Prva ima podatke naožene v S3, druga pa v HDFS. Časovna primerjava je smiselna med lokalno gručo treh vozlišč in gručo treh srednjih vozlišč v oblaku s podatki v HDFS.

Gruča	Tip	Čas obdelave
3 srednja vozlišča (HDFS)	lokalna	45 min 57 sek
3 srednja vozlišča (S3)	oblak	100 min
3 srednja vozlišča (HDFS)	oblak	47 min 25 sek

Tabela 10: Primerjava lokalne gruče in gruč v oblaku

Iz tabele 10 je razvidno, da je najhitrejša gruča med primerjanimi lokalna. Logično bi bilo, če bi sklepali po razpoložljivosti pomnilnika, da bi bila najhitrejša gruča iz oblaka z načinom shranjevanja v HDFS. Lokalna gruča ima na voljo 5632MB pomnilnika (glej poglavje 3.5) in štiri procesorska jedra. Gruča v oblaku ima na voljo 6GB pomnilnika in tri procesorska jedra (glej specifikacijo strojne opreme vozlišč v poglavju 4.2). Kljub temu, da ima lokalna gruča manj pomnilnika ima eno procesorko jedro več, kar po moji oceni pripomore k hitrejši izvedbi obdelave. Čeprav je lokalna gruča hitrejša, je potrebno več dela pri postavitvi gruče zato je v tem primeru bolje uporabiti oblak.

5.3 Cenovna primerjava med izvajanjem ene obdelave v oblaku in mojem lokalnem okolju

V primeru poganjanja na lokalni infrastrukturi je potrebno upoštevati stroške nakupa vozlišč, učenje namestitve in morebitne optimizacije Hadoop, postavitev gruče ter vzdrževanje. Če nas daljši čas obdelave ne moti lahko prihranimo pri stroških nakupa vozlišč, vzdrževanja, postavitvi gruče in Hadoop postavimo na osebnem računalniku. Vendar je v takem primeru vsekakor bolje uporabiti oblak saj se ni potrebno posvečati namestitvi.

Na spletnem bolšjem trgu Bolha.si sem našel rabljen strežnik, ki je po strojni opremi primerljiv z mojo lokalno infrastrukturo in gruči treh srednjih vozlišč v oblaku.

To je HP ProLiant ML110 G7 s specifikacijami:

- Intel Xeon E3-1220 (3.10GHz)
- 2 GB pomnilnika
- 250 GB trdega diska

Cena takega rabljenega strežnika je približno 350 EUR oz. po dogovoru. Če zraven dodamo še dva, dobimo gručo treh vozlišč za katero odštejemo okoli 1.050 EUR. V tabeli 11 so prikazane aktivnosti ene obdelave v lokalni gruči vozlišč z zgornjimi specifikacijami. Kljub

temu, da primerjam eno obdelavo je jasno, da lahko v kupljeni lokalni infrastrukturi za isto ceno poganjamo poljubno število obdelav (do odpovedi strojne opreme).

Aktivnost	Čas (h)	Cena (EUR)
Lokalna infrastruktura		
Nakup infrastrukture	~ 2	~ 1.110
Namestitev gruče	~ 4	~ 140
Namestitev Hadoop	~ 8	~ 280
Skupni stroški (glej tabelo 6)	~ 23	~ 805
Vzdrževanje	~ 8	~ 280
SKUPAJ:	**~ 45**	**~ 2.615**
Vzdrževanje	~ 4	~ 140
Elektrika		~ 20
MESEČNI STROŠEK:		**~ 160**

Tabela 11: Prikaz aktivnosti ene obdelave v lokalnem okolju po času in stroških

V primeru poganjanja ene obdelave v oblaku je potrebno poleg skupnih stroškov upoštevati najem gruče (glej tabelo 9) ter čas porabljen za najem preko vmesnika.

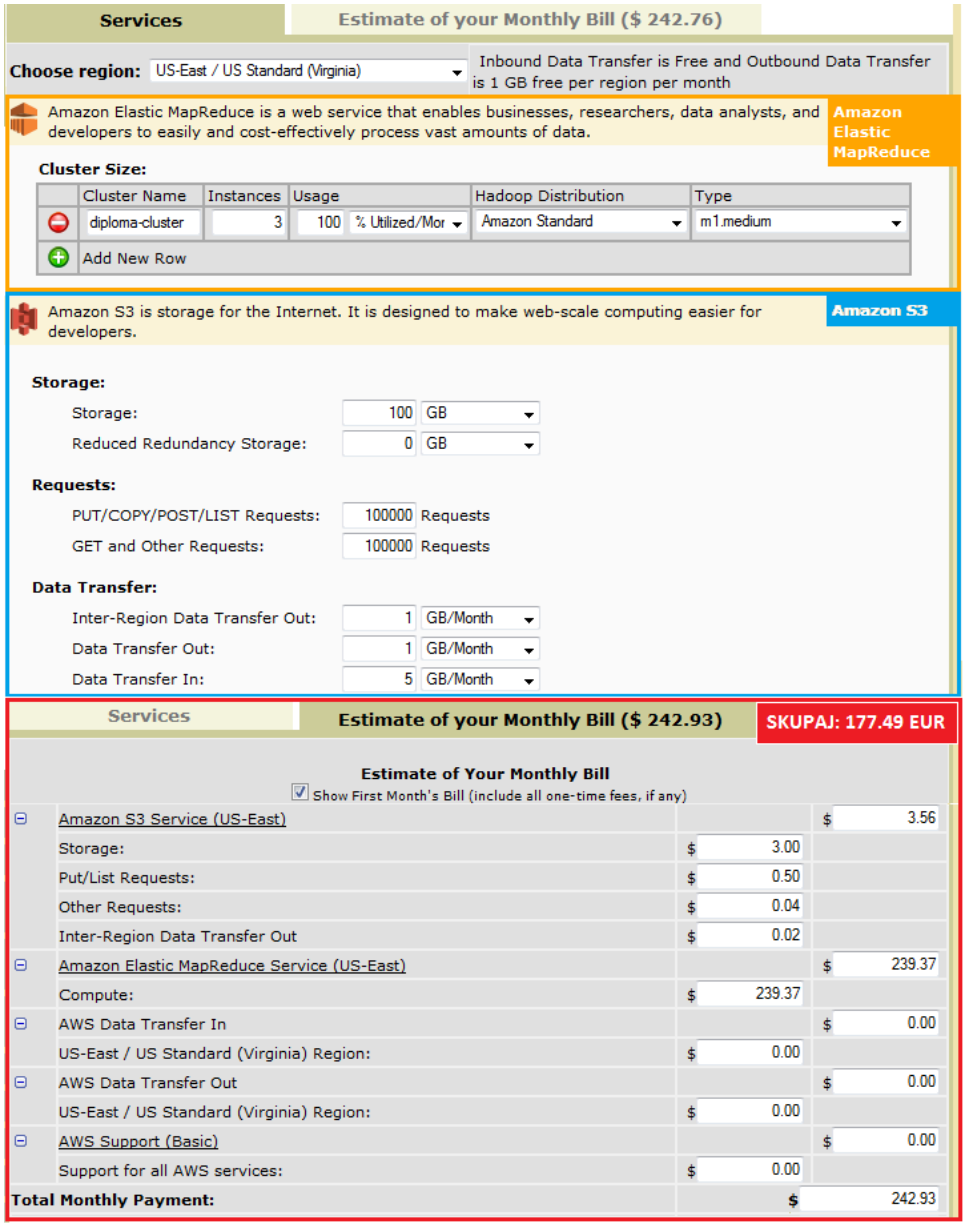
Aktivnost	čas (h)	cena (EUR)
Oblak (gruča treh srednjih vozlišč s podatki v HDFS)		
Najem infrastrukture preko vmesnika	~ 1	~ 35
Poganjanje na gruči treh srednje velikih vozlišč	~ 2	~ 0,654
Prenos podatkov v oblak	~ 5	0
Prenos podatkov iz oblaka	/	~ 0,0001
Skupni stroški (glej tabelo 6)	~ 23	~ 805
Hranjenje podatkov v oblaku (mesečni strošek)	/	~ 0,77
SKUPAJ:	**~ 31**	**~ 841,424**
MESEČNI STROŠEK:		**~ 0,77**

Tabela 12: Prikaz aktivnosti ene obdelave po času in stroških [2,3]

V tabeli 12 je razvidno, da je največji strošek poganjanja obdelav v oblaku razvojni cikel obdelave. Ponovno se oblak izkaže kot najbolj ugodna rešitev za poganjanje ene obdelave.

5.4 Cenovna primerjava med najemom gruče treh vozlišč za konstantno izvajanje obdelav v oblaku in nakupom primerljive infrastrukture za lokalno uporabo

Predhodne primerjave so temeljile na dejstvu, da poganjamo eno obdelavo. V primeru, da moramo imeti gručo na voljo vse dni v mesecu se stroški v oblaku močno povečajo medtem ko stroški nakupa lokalne infrastrukture ostanejo enaki (dodaten strošek je mesečno redno vzdrževanje). Amazon na svojih spletnih straneh ponuja enostaven interaktivni kalkulator, ki pomaga izračunati predvideno mesečno porabo za izbrane storitve (glej sliko 30).



Slika 30: Izračun mesečnega najema gruče treh srednjih vozlišč v oblaku [16]

Izračun na sliki 30 prikazuje mesečni strošek najema gruče treh srednjih vozlišč in uporabe 100GB porstora na S3, ki znaša 177,49 EUR. Iz tega sledi, da se nam strošek nakupa lokalne infrastrukture (glej nakup infrastrukture v tabeli 11), ki je primerljiva z gručo treh srednjih vozlišč v oblaku povrne v letu in pol konstantne uporabe pri predpostavki, da nimamo dodatnih stroškov odpovedi strojne opreme lokalne gruče. Tabela 13 prikazuje primerjavo nakupa lokalne gruče in najema v oblaku za obdobje enega leta.

Strošek	Čas (h)	Cena (EUR)
3 srednja vozlišča v oblaku		
Letni najem		2.129,88
Skupaj letno:		**2.129,88**
3 srednja vozlišča lokalno		
Nakup infrastrukture	~ 2	~ 1.110
Namestitev gruče		~ 140
Namestitev Hadoop		~ 280
Vzdrževanje	~ 48	~ 1.200
Elektrika		~ 240
Skupaj letno:	**~ 50**	**~ 2.970,00**

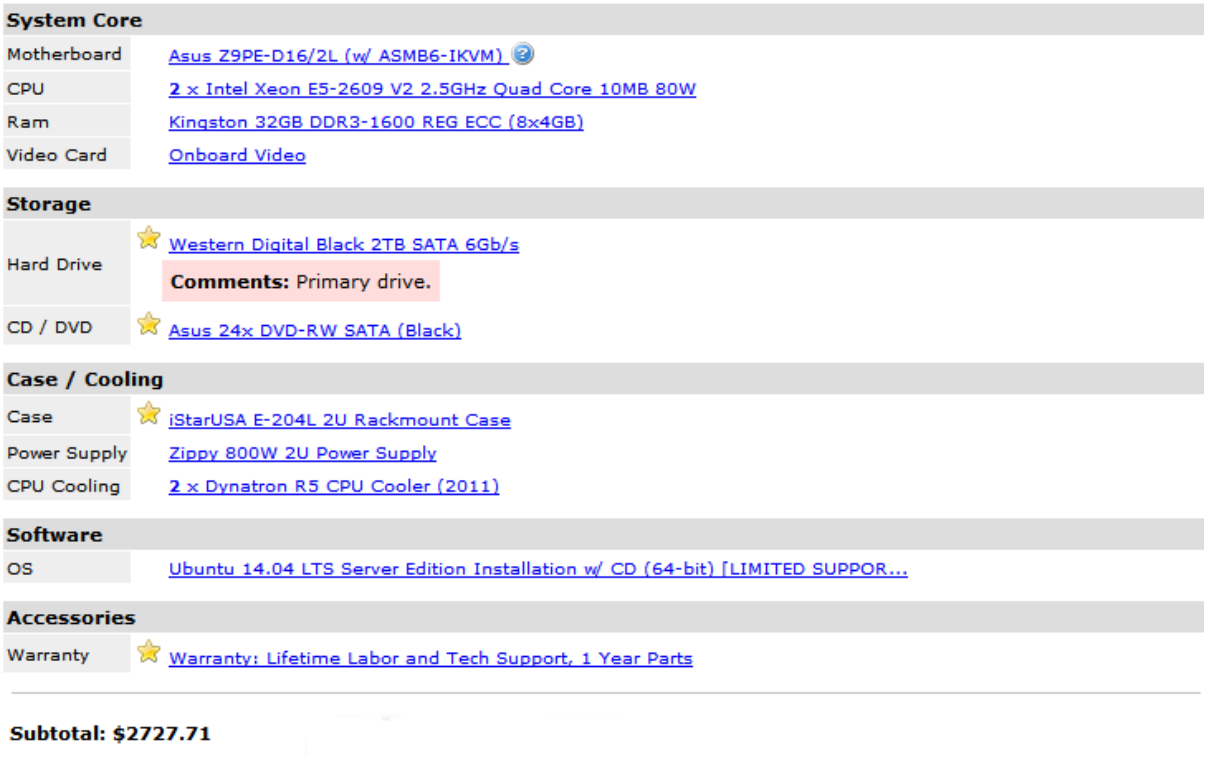
Tabela 13: Primerjava nakupa lokalne gruče in najema gruče treh srednjih vozlišč v oblaku

Srednja vozlišča so najmanjša vozlišča, ki jih je mogoče uporabiti v povezavi z Amazon EMR. Tudi v tem primeru je boljši oblak saj nikoli ne vemo kdaj bo prišlo do odpovedi strojne opreme na lokalni gruči. Za odpoved na lokalni gruči je potrebno skrbeti in strošek odpovedi ni zanemarljiv. Odpovedi v Amazon gručah so pravtako mogoče. Amazon zagotavlja 99,9% dosegljivost sistema in celovitost podatkov neglede na odpovedi.

5.5 Cenovna primerjava med najemom gruče 10 srednjih vozlišč za konstantno izvajanje obdeav v oblaku in nakupom primerljive infrastrukture za lokalno uporabo

V prejšnjem podpoglavju sem primerjal dve zelo majhni gruči, ki se najverjetneje v svetu velike količine podatkov ne bodo pojavile prav pogosto. Kot zanimivost si poglejmo stroške nakupa večje lokalne gruče 10 vozlišč in jo primerjajmo z gručo 10 zelo velikih vozlišč v oblaku.

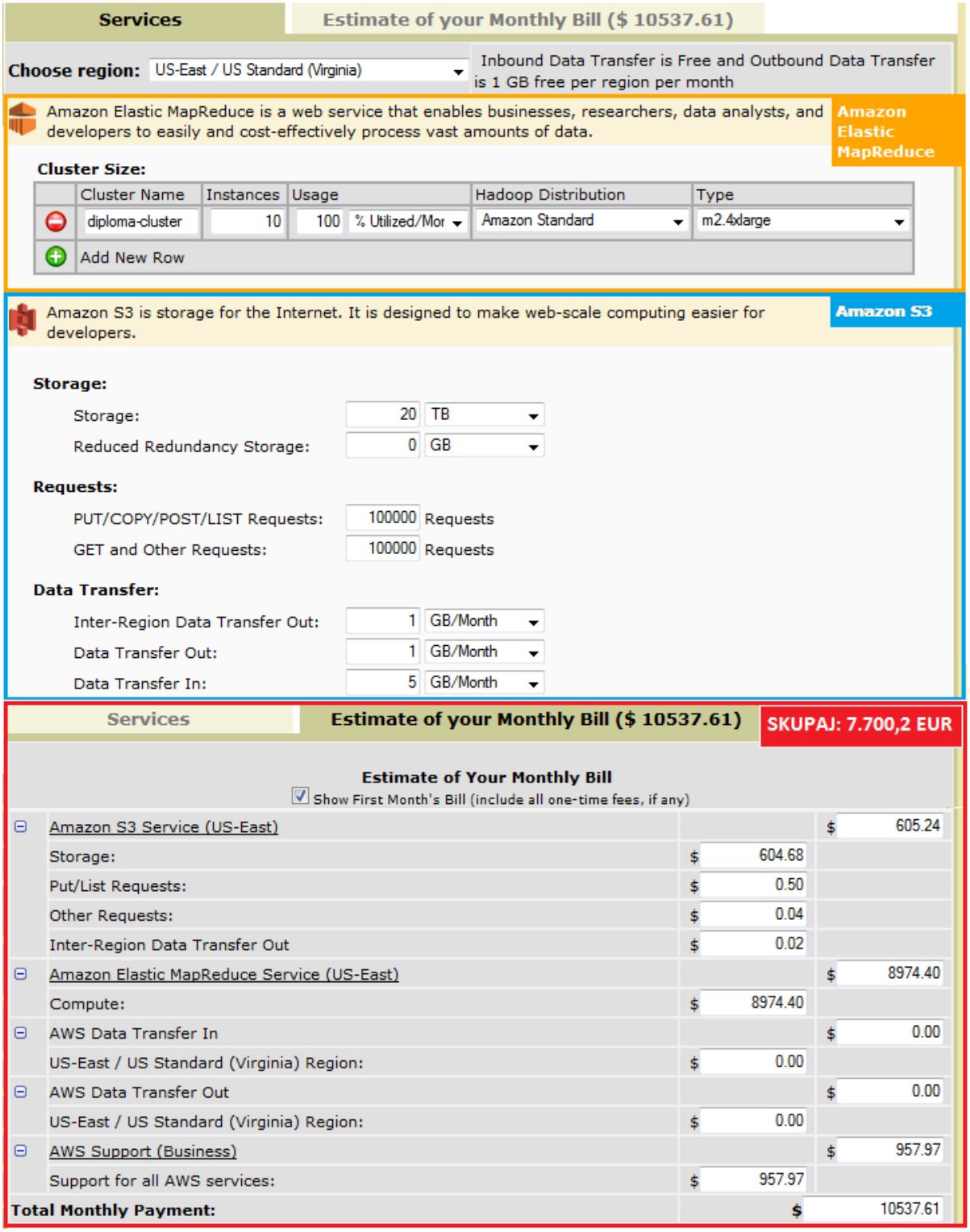
Slika 31 prikazuje spisek komponent in ceno enega vozlišča, ki sem ga ustvaril sam s pomočjo spletne strani podjetja Puget Systems [14].



Slika 31: Prikaz komponent in cena enega vozlišča 2.002,43 EUR [14]

Cena gruče desetih vozlišč, ki jo sami sestavimo in je primerljiva z zmogljivostjo deset zelo zmogljivih vozlišč v oblaku, je približno 25.000 EUR. V ceno sem poleg strojne opreme vključil še delo infrastrukturnih specialistov, namestitev gruče in komponent Hadoop ter stikalo za povezavo vozlišč. Gruča lahko hrani nekaj manj kot 20 TB. Za vzdrževanje take gruče bi najbrž porabili okoli 500 EUR na mesec. Poleg tega, bi bila mesečna poraba elektrike približno 80 EUR.

Ta gruča je dostopna vse dni v letu in po nakupu nam edini strošek predstavljajo elektrika, vzdrževanje in razvoj obdelav. Poglejmo si še strošek mesečnega najema Amazon EMR gruče deset zelo zmogljivih vozlišč, ki je dostopna vse dni v letu. Cena take gruče je visoka. Če vzamemo ceno za eno uro poganjanja na gruči deset zelo velikih vozlišč in jo enostavno pomnožimo s številom ur v mesecu, dobimo 6.458,4 EUR, temu dodamo še stroške S3, ki znašajo za 20 TB približno 440 EUR na mesec in dobimo 6.898,4 EUR. Ta izračun ni povsem natančen, pa si pomagajmo z Amazonovim spletnim kalkulatorjem, ki nam bo dal bolj natančen podatek (glej sliko 32).



Slika 32: Izračun mesečnega najema gruče treh zelo velikih vozlišč v oblaku [16]

Za 20 TB prostora na S3 bi mesečno odšteli 442.76 EUR. Skupaj z EMR pa kar 7.700,2 EUR. Vidimo, da pri zahtevnejši infrastrukturi v oblaku in potrebi po hranjenju velike količine

podatkov cena hitro naraste. V tem primeru se nam investicija v lokalno gručo po moji oceni povrne že po štirih mesecih konstantnega poganjanja obdelav.

Za večja gruče, bi se sam sigurno odločil za lokalno gručo. Čeprav mogoče predstavlja nekoliko več dela pri postavitvi in vzdrževanju je vseeno bolje imeti podatke pri sebi.

Poglavje 6 Povzetek in sklepne ugotovitve

V sklopu diplomske naloge sem dosegel vse cilje, ki so bili zastavljeni v uvodu.

6.1 Povzetek prednosti in slabosti poganjanja obdelav v oblaku

Glavne prednosti poganjanja obdelav v oblaku so po moji oceni naslednje:

- dobro in cenovno ugodno za poganjanje namenskih obdelav,
- enostavno za prikaz delovanja prve verzije programske opreme, brez nakupa infrastrukture,
- razširjanje gruč, brez posegov administratorjev in nakupa novih vozlišč,
- primerno za testiranje pravilnosti obdelav,
- ni potrebno poznavanje arhitekture in namestitve sistema,
- ni vzdrževanja strojne opreme,
- ni skrbi ob odpovedi strojne opreme.

Slabosti:

- podatke zaupamo tretji osebi,
- če so podatki naloženi v S3 in ne v HDFS izgubimo funkcionalnost poganjanja obdelav v bližini podatkov,
- podatke je potrebno prej naložiti v oblak,
- težja integracija z drugimi sistemi, ki niso v oblaku,
- izpad internetnega omrežja iz kakršnegakoli razloga,
- povezovanje na oddaljene strežnike preko SSH in SFTP protokolov ni trivialno.

6.2 Povzetek prednosti in slabosti poganjanja obdelav na lokalni gruči

Prednosti poganjanja obdelav na lokalni gruči:

- lokalnost podatkov,
- poganjanje obdelav na vozliščih kjer se nahajajo podatki, ker uporablja HDFS,
- varnost podatkov,
- stalna razpoložljivost gruče,
- boljša rešitev za sisteme, kjer se obdelave konstantno poganjajo,
- lahka integracija z drugimi lokalnimi in občutljivimi podatki (npr. finančni izkazi).

Slabosti:

- strošek nakupa, uporabe (npr. hlajenje, elektrika) in predvsem vzdrževanja gruče,
- namestitev in optimizacija gruče,
- obvladovanje odpovedi strojne opreme,
- dodajanje vozlišč je enostavno, vendar zahteva nakup nove opreme in plačilo za namestitev.

6.3 Povzetek diplomske naloge

Opisal sem najbolj razširjen pojem pri obdelavi velike količine podatkov Hadoop, predstavil MapReduce programsko ogrodje in njegovo delovanje. Razvil sem enostavno aplikacijo MapReduce in skripto Pig. Postavil gručo Hadoop na lokalni infrastrukturi. Odprl sem račun na Amazon in aplikacijo MapReduce pognal v oblaku in na lokalni infrastrukturi ter prikazal rezultate. Predstavil sem časovni in cenovni vidik implementacije ene obdelave in primerjal poganjanje med oblakom in lokalno infrastrukturo. Za zaključek primerjave sem podal še primer dobre lokalne infrastrukture in jo primerjal s primerljivimi vozlišči v oblaku.

Rezultati so bili pričakovani. Na moji lokalni infrastrukturi se vse skupaj ni dobro obnašalo, saj sem uporabil nekoliko zastarele prenosne računalnike in virtualizacijo. Oblak mi je omogočil test MapReduce na dobri infrastrukturi. Obdelava je v najboljši uporabljeni gruči z

uporabo datotečnega sistema HDFS trajala 1 minuto in 36 sekund. To je zelo dober čas za obdelavo 35 GB podatkov (291310268 vhodnih vrstic). Enaka gruča je z uporabo S3 datotečnega sistema obdelavo zaključila v 3 minutah. Verjamem, da bi z nekoliko truda pri optimizaciji Java programske kode lahko ta čas še izboljšal. Uporaba S3 datotečnega sistema je zelo enostavna in najbolj primerna za obdelave, ki jih poganjajo manj tehnični uporabniki, ki nimajo želje po poznavanju Hadoop ukazov in poizvedbe pišejo s pomočjo Pig Latin. Medtem, ko HDFS zagotavlja enake čase obdelav, kot jih bomo srečevali v lokalnem sistemu, kar je zelo priročno za testiranje časov izvajanja, ko so le-ti pomembni.

Oblak bi ocenil kot zelo dobro rešitev za poganjanje namenskih obdelav, ki odgovarjajo na specifična vprašanja o podatkih. Prav tako je oblak dobra rešitev za postavljanje pilotov in testnih primerov, preden se lotimo implementacije ali aplikacijo poženemo na lokalni gruči. V tem primeru moramo podatke samo naložiti v oblak in napisati aplikacijo MapReduce. Lokalna infrastruktura je boljša v primeru, da želimo Hadoop uporabljati kot orodje za konstantno poganjanje obdelav in omogočiti stalno dosegljivost sistema uporabnikom (analitikom). V takem primeru se investicija v nakup lokalne gruče že v kratkem času povrne.

Diploma daje bralcu celoten vplogled v svet osnovnih komponent Hadoop. Od razvoja MapReduce do postavitve lokalne gruče in prikaza uporabe oblaka. Bralec si lahko z diplomo pomaga postaviti lokalne gruče, Java kodo ali skripto Pig pa uporabi za ogrodje svojege prve obdelave.

6.4 Sklepne misli

V tem diplomskem delu se nisem dotaknil vseh podprojektov Hadoop. Menim, da so vsi prvonivojski podprojekti zelo dobri in bralcu predlagam pregled njihovih funkcionalnosti. Za bralce iz relacijskega sveta podatkovnih baz bi lahko bil zanimiv produkt Hive, ki podatke uredi v strukturirane tabele, po katerih lahko poizvedujemo z jezikom podobnim SQL (*angl. Structured Query Language*). Poleg projektov, ki spadajo pod Hadoop, obstajajo tudi različne distribucije Hadoop, ki nam olajšajo namestitev v gruči in pomagajo pri optimizaciji in integraciji. Dobra primera sta Cloudera in MapR.

Prav tako bi bralcem predlagal, da si ogledajo nov produkt Apache Spark, ki se hvali z 10 % hitrejšimi obdelavami kot Hadoop na disku in 100 % hitrejšimi v pomnilniku.

DODATEK A: KODA

- BMapReduce.java

```
package si.bron;
import java.net.URI;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class BMapReduce extends Configured implements Tool{

   @Override
   public int run(String[] arg0) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf,"BMapReduce");
      job.addCacheFile(new URI(arg0[2]));
      job.setJarByClass(BMapReduce.class);
      job.setJobName("Avg Temperature MapReduce V3");
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(DoubleWritable.class);
      job.setMapperClass(BMapper.class);
      job.setCombinerClass(BCombiner.class);
      job.setReducerClass(BReducer.class);
      job.setNumReduceTasks(1);
      FileInputFormat.setInputDirRecursive(job, true);
```

```
        FileInputFormat.setInputPaths(job, new Path(arg0[0]));
        FileOutputFormat.setOutputPath(job, new Path(arg0[1]));
        job.waitForCompletion(true);
        return 0;
    }
        public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new BMapReduce(), args);
        System.exit(res);
    }
}
```

- BMapper.java

```
package si.bron;
import java.io.*;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class BMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {

    public enum Temperature {
        MISSING_TEMP,
        ARRAY_OUT_OF_BOUNDS,
        UNKNOWN
    }

    public BMapper() {
    }

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        Double dryBulbCelsius = new Double(0.0);
        String year = "";
        String WBAN = "";
        String[] splitter = value.toString().split(",");
            try {
```

```
            dryBulbCelsius = Double.parseDouble(splitter[12]);
            year = splitter[1].substring(0, 4);
            WBAN = splitter[0];
            context.write(new Text(year +"\t"+ WBAN),new
DoubleWritable(dryBulbCelsius));
         }catch(Exception ex) {
            if(ex instanceof NumberFormatException) {
               context.getCounter(Temperature.MISSING_TEMP).increment(1);
            }else if(ex instanceof ArrayIndexOutOfBoundsException) {
               context.getCounter(Temperature.ARRAY_OUT_OF_BOUNDS).increment(1);
            }else {
               context.getCounter(Temperature.UNKNOWN).increment(1);
            }
         }
      }
}
```

- BReducer.java

```
package si.bron;
import java.io.FileNotFoundException;
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import si.bron.metadata.StationMap;

public class BReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
   private StationMap stationMap;

   public BReducer() {
   }

   private Double avg(Iterable<DoubleWritable> values) {
      Double sum = new Double(0.0);
      int count = 0;
      for(DoubleWritable val : values) {
         sum+=val.get();
```

```
            count++;
        }
        return sum / count;
    }
    @Override
    protected void setup(Context context) throws IOException, InterruptedException{
        @SuppressWarnings("deprecation")
        Path[] dcPath = context.getLocalCacheFiles();
         if(dcPath.length == 0) {
            throw new FileNotFoundException("File not found");
        }
        stationMap = new StationMap(dcPath[0].toString());
    }

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values,Context context)
            throws IOException, InterruptedException {
        String[] splitKey = key.toString().split("\t");
        Text newKey = new Text(key +"\t"
+stationMap.getStationSideDataByKey(splitKey[1]));
        context.write(newKey, new DoubleWritable(avg(values)));
    }
}
```

- BCombiner.java

```
package si.bron;
import java.io.IOException;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class BCombiner extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    public BCombiner() {
    }

    private Double avg(Iterable<DoubleWritable> values) {
        Double sum = new Double(0.0);
```

```
        int count = 0;
        for(DoubleWritable val : values) {
            sum+=val.get();
            count++;
            //context.progress();
        }
        return sum / count;
    }

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values,Context context)
            throws IOException, InterruptedException {
       context.write(key, new DoubleWritable(avg(values)));
    }
}
```

- Station.java

```
package si.bron.metadata;

public class Station {
    private String wban;
    private String location;
    private String latitude;
    private String longitude;

    public Station(String stationRecord) {
        setup(stationRecord);
    }
    private void setup (String stationRecord) {
        String[] splitRecord = stationRecord.split(",");
        setWban(splitRecord[0]);
        setLocation(splitRecord[1]);
        setLatitude(splitRecord[2]);
        setLongitude(splitRecord[3]);
    }
    public String getWban() {
        return wban;
```

```
    }
    public void setWban(String wban) {
        this.wban = wban;
    }
    public String getLocation() {
        return location;
    }
    public void setLocation(String location) {
        this.location = location;
    }
    public String getLatitude() {
        return latitude;
    }
    public void setLatitude(String latitude) {
        this.latitude = latitude;
    }
    public String getLongitude() {
        return longitude;
    }
    public void setLongitude(String longitude) {
        this.longitude = longitude;
    }
}
```

- StationMap.java

```
package si.bron;
import java.io.IOException;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class BCombiner extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
    public BCombiner() {
        // TODO Auto-generated constructor stub
    }
    private Double avg(Iterable<DoubleWritable> values) {
        Double sum = new Double(0.0);
        int count = 0;
```

```
        for(DoubleWritable val : values) {
            sum+=val.get();
            count++;
            //context.progress();
        }
        return sum / count;
    }
    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values,Context context)
            throws IOException, InterruptedException {
                context.write(key, new DoubleWritable(avg(values)));

    }
}
```

- BMapReduceTest.java

```
public class BMapReduceTest {
    @Test
    public void basicMapperInputTest() throws IOException {
        Text value = new Text ("03011,20100201,0013,0,CLR, ,10.00, , , ,19, ,-7.0, ,17, ,-8.1,
,14, ,-10.0, , 81, , 3, ,180, , , ,21.33, , , , , ,M, ,AA, , , ,29.94");
        Pair<LongWritable, Text> inputRecord = new Pair<LongWritable, Text>(new
LongWritable(1),value);
        new MapDriver<LongWritable, Text, Text, DoubleWritable>()
        .withMapper(new BMapper())
        .withInput(inputRecord )
        .withOutput(new Text("2010"),new DoubleWritable(-7.0))
        .runTest();
    }
    @Test
    public void basicMapperInputTestArrayIndexOutOfBounds() throws IOException {
        Text value = new Text ("03011,20100201,0013,0,CLR, ,10.00, , , ,19");
        Pair<LongWritable, Text> inputRecord = new Pair<LongWritable, Text>(new
LongWritable(1),value);
        new MapDriver<LongWritable, Text, Text, DoubleWritable>()
        .withMapper(new BMapper())
        .withInput(inputRecord )
        .runTest();
```

```
    }
    @Test
    public void basicMapperInputTestHeader() throws IOException {
        Text value = new Text
("WBAN,Date,Time,StationType,SkyCondition,SkyConditionFlag,Visibility,VisibilityFlag,
WeatherType,WeatherTypeFlag,DryBulbFarenheit,DryBulbFarenheitFlag,DryBulbCelsius,Dr
yBulbCelsiusFlag,WetBulbFarenheit,WetBulbFarenheitFlag,WetBulbCelsius,WetBulbCelsiu
sFlag,DewPointFarenheit,DewPointFarenheitFlag,DewPointCelsius,DewPointCelsiusFlag,Re
lativeHumidity,RelativeHumidityFlag,WindSpeed,WindSpeedFlag,WindDirection,WindDire
ctionFlag,ValueForWindCharacter,ValueForWindCharacterFlag,StationPressure,StationPress
ureFlag,PressureTendency,PressureTendencyFlag,PressureChange,PressureChangeFlag,SeaL
evelPressure,SeaLevelPressureFlag,RecordType,RecordTypeFlag,HourlyPrecip,HourlyPrecip
Flag,Altimeter,AltimeterFlag");
        Pair<LongWritable, Text> inputRecord = new Pair<LongWritable, Text>(new
LongWritable(1),value);
        new MapDriver<LongWritable, Text, Text, DoubleWritable>()
        .withMapper(new BMapper())
        .withInput(inputRecord)
        .runTest();
    }
    @Test
    public void basicMapperInputMissing() throws IOException {
        Text value = new Text ("03011,20100225,0712,0,BKN016 OVC026, , 0.75, ,-SN, ,M,
,M, ,M, ,M, ,M, ,M, ,M, , 0, ,000, , , ,21.33, , , , , ,M, ,AA, , , ,29.93,");
        Pair<LongWritable, Text> inputRecord = new Pair<LongWritable, Text>(new
LongWritable(1),value);
        new MapDriver<LongWritable, Text, Text, DoubleWritable>()
        .withMapper(new BMapper())
        .withInput(inputRecord)
        .withCounter(BMapper.Temperature.MISSING_TEMP, 1)
        .runTest();
    }
    @Test
    public void basicReducerInputTest() throws IOException {
        List<DoubleWritable> temperatureList = Arrays.asList(new DoubleWritable(-7.0),new
DoubleWritable(5.0));
        Text key = new Text("03011"+"\t"+"2010");
```

```
        Text outputKEy = new Text("03011"+"\t"+"2010"+"\t"+"TELLURIDE REGIONAL
AIRPORT"+"\t"+"37.954"+"\t"+"-107.901");
        Pair<Text, List<DoubleWritable>> inputRecord = new Pair<Text,
List<DoubleWritable>>(key,temperatureList);
        new ReduceDriver<Text, DoubleWritable, Text, DoubleWritable>()
        .withReducer(new BReducer())
        .withInput(inputRecord)
        .withCacheFile("/home/hsingle/stations.txt")
        .withOutput(outputKEy,new DoubleWritable(-1))
        .runTest();
    }
```

DODATEK B: Nastavitve lokalne gruče

Nastavitve na hsingle (glavno vozlišče in upravitelj virov):

- core-site.xml:

```
<configuration>
   <property>
         <name>fs.defaultFS</name>
         <value>hdfs://hsingle:10001</value>
         <description>Globalna nastavitev tipa podatkovnega sistema. Privzeto HDFS, možnosti so še npr. Amazon S3, lokalni podatkovni sistem operacijskega sistema (neporazdeljen podatkovni sistem). Hdfs:// pomeni, da gre za HDFS podatkovni sistem, potrebno je še dodati ime strežnika in privzeta vrata za komunikacijo z drugimi procesi. </description>
   </property>
   <property>
         <name>hadoop.tmp.dir</name>
         <value>/usr/local/hadoop/tmp</value>
         <description>Nastavitev začasne mape za HDFS</description>
   </property>
</configuration>
```

- hdfs-site.xml

```
<configuration>
   <property>
         <name>dfs.replication</name>
         <value>3</value>
         <description>Replikacijski faktor za bloke HDFS. Privzeta vrednost je 2</description>
   </property>
   <property>
         <name>dfs.datanode.address</name>
```

<value>hsingle:50010</value>
<description>Nastavitev DataNode privzetih vrat za komunikacijo med procesi. S tem tudi povemo, da je na strežniku poleg NameNode tudi DataNode</description>
</property>
<property>
<name>dfs.datanode.http.address</name>
<value>hsingle:50075</value>
<description>Nastavitev http naslova za podatkovno vozlišče. Na tem naslovu bodo prikazane statistike podatkovnih vozlišč, ki jih lahko uporabnik pregleduje v spletnem brskalniku.</description>
</property>
</configuration>

- mapred-site.xml

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
    <description>Povemo, da za upravljanje MapReduce gruča uporabi YARN</description>
  </property>
</configuration>
```

- yarn-site.xml

```
<configuration>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
     <value>hsingle:8031</value>
    <description>Nastavitev vrat za sledilnika računalniških virov. Komunikacija med upraviteljem z viri in upraviteljem vozlišča </description>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>hsingle:8088</value>
    <description>HTTP naslov za pregledovanje obdelav.</description>
```

```
    </property>
    <property>
        <name>yarn.resourcemanager.admin.address</name>
        <value>hsingle:8033</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>hsingle:8030</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>hsingle:8032</value>
        <description>Naslov, na katerem je upravitelj virov. Uporablja se za
    komunikacijo med upraviteljem virov in upraviteljem vozlišča.</description>
    </property>
    <property>
        <name>yarn.nodemanager.address</name>
        <value>hsingle:8050</value>
        <description>Ker imamo na vozlišču upravitelja z viri in upravitelja vozlišča,
    moramo definirati vrata na katerih je dostopen upravitelj vozliča, ki morajo biti
    različna od vrat upravitelja virov.</description>
    </property>
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>3072</value>
    <description>Nastavitev porabe spomina, ki jo lahko upravitelj vozlišča uporabi na
hsingle v MB</description>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
        <description>Specifikacija zunanjega procesa za MapReduce fazo
    premetavanja </description>
    </property>
  </configuration>
```

- slaves

  hsingle
  hsingle2
  hsingle3

  V nastavitvi slaves povemo, na katerih računalnikih so podatkovna vozlišča.

- master

  hsingle

  V nastavitvi master povemo, da je glavno vozlišče računalnik hsingle.

Nastavitve na hsingle2 in hsingle3 (podatkovno in upraviteljsko vozlišče)

- hdfs-site.xml

```
<configuration>
  <property>
        <name>dfs.replication</name>
        <value>3</value>
        <description>Replikacijski faktor za bloke HDFS. Privzeta vrednost je 2</description>
  </property>
  <property>
        <name>dfs.datanode.address</name>
        <value>hsingle2:50010</value>
        <description>Nastavitev podatkovnega vozlišča na hsingle2</description>
  </property>
  <property>
        <name>dfs.datanode.http.address</name>
        <value>hsingle2:50075</value>
        <description>Nastavitev http naslova za podatkovno vozlišče hsingle2</description>
  </property>
</configuration>
```

- yarn-site.xml

```
    <configuration>
      <property>
            <name>yarn.resourcemanager.address</name>
            <value>hsingle:8032</value>
            <description>Naslov, na katerem je upravitelj virov. Uporablja se za
      komunikacijo med upraviteljem virov in upraviteljem vozlišča</description>
      </property>
      <property>
            <name>yarn.nodemanager.address</name>
            <value>hsingle2:8050</value>
            <description>upravitelj vozlišč vrata za hsingle2</description>
      </property>
      <property>
            <name>yarn.nodemanager.resource.memory-mb</name>
            <value>1536</value>
            <description>Nastavitev porabe spomina za upravitelje vozlišča na hsingle2
      1536 MB na hsingle3 1024 MB</description>
      </property>
<property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
            <description>Specifikacija zunanjega procesa za MapReduce fazo
      premetavanja </description>
      </property>
 </configuration>
```

Kazalo slik

Kazalo tabel

Literatura in viri

Literatura

[1] T. White, *Hadoop: The Definitive Guide*, Third Edition, California: O'Reilly Media, 2012

Spletni viri

[2] Amazon EC2, cenik najema infrastrukture v oblaku. Dostopno 30.6.2014 na: http://aws.amazon.com/ec2/pricing/

[3] Amazon EMR, cenik najema infrastrukture v oblaku. Dostopno 30.6.2014 na: http://aws.amazon.com/elasticmapreduce/pricing/

[4] Amazon S3, Hadoop Wiki. Dostopno 30.6.2014 na: https://wiki.apache.org/hadoop/AmazonS3

[5] Apache Hadoop NextGen MapReduce (YARN) (2014). Dostopno 30.6.2014 na: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[6] C. Versace (2014), *The Big Deal About Big Data And What It Means For IT and You*. Dostopno 30.6.2014 na: http://www.forbes.com/sites/chrisversace/2014/01/28/the-big-deal-about-big-data-and-what-it-means-for-it-and-you/

[7] D. Sullivan (2014), *Getting Started with Hadoop 2.0.* Dostopno 30.6.2014 na: http://www.tomsitpro.com/articles/hadoop-2-vs-1,2-718.html

[8] HDFS Architecture Guide (2013). Dostopno 30.6.2014 na: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[9] J. Dean, S. Ghemawat (2004), *MapReduce: Simplied Data Processing on Large Clusters*. Dostopno 30.6.2014 na: http://static.googleusercontent.com/media/research.google.com/sl//archive/mapreduce-osdi04.pdf

[10] J. Gantz, D. Rainsel (2012), IDC, *The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*. Dostopno 30.6.2014 na: http://idcdocserv.com/1414

[11] M. Ferle (2013), *Hadoop in MapReduce*. Dostopno 30.6.2014 na: http://www.monitorpro.si/156498/praksa/hadoop-in-mapreduce/

[12] N. Bandugula (2012), *Breaking the Minute Barrier for TeraSort*. Dostopno 30.6.2014 na: http://www.wired.com/2012/11/breaking-the-minute-barrier-for-terasort/

[13] Podatki o vremenu, *National Climatic Data Center*. Dostopno 30.6.2014 na: http://www.ncdc.noaa.gov/data-access/quick-links#loc-clim

[14] Puget Systems, Puget Summit Servers. Spletna stran za sestavljanje strežnikov. Dostopno 30.6.2014 na: https://www.pugetsystems.com

[15] S. Ghemawat, H. Gobioff, S. T. Leung (2003), *The Google File System*. Dostopno 30.6.2014 na: http://static.googleusercontent.com/media/research.google.com/sl//archive/gfs-sosp2003.pdf

[16] Stroški najema infrastrukture, Amazon Web Services Simple Monthly Calculator. Dostopno 30.6.2014 na: http://calculator.s3.amazonaws.com/index.html